# Where Do Open Source Requirements Come From (And What Should We Do About It)?

A Position Paper for the Second ICSE Workshop on Open Source Software Engineering

Bart Massey
Computer Science Department
Portland State University
P.O. Box 751 M.S. CMPS
Portland, Oregon USA 97207–0751
bart@cs.pdx.edu

## ABSTRACT

The collection and specification of software requirements is one of the most intense areas of software engineering research. This makes it a natural area to explore when considering open-source software. In this paper, I argue that the sources of open-source software requirements differ in some important respects from the sources of commercial software project requirements. This has some interesting implications for both open-source and commercial development. The collection and specification of software requirements is one of the most intense areas of software engineering research. This makes it a natural area to explore when considering open-source software. In this paper, I argue that the sources of open-source software requirements differ in some important respects from the sources of commercial software project requirements. This has some interesting implications for both open-source and commercial development.

## Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications

## Keywords

Free Software, Open Source, Requirements

## 1. THE SOFTWARE DEVELOPMENT HERITAGE

Throughout the brief history of software development, the process of actually constructing computer programs has remained surprisingly constant. Fred Brooks' classic *The Mythical Man-Month* [2] was re-issued in a 20th Anniversary printing recently: it is now more than 25 years old. In spite of this, it describes an "industry standard" software development process fundamentally similar to current practice: the most significant change being the widespread adoption of higher-level programming languages.

In contrast, the motivations for constructing computer programs and the functionality required from those programs have varied widely through time and across industries. It might be argued that most of the significant advances in software process have been less about determining how to build software than about determining what software to build.

The early days of software development were notable for a carefree attitude toward the commercial potential of software. Simply put, software was built to meet "obvious" requirements, and disseminated, often in source form, in the hopes that it might be useful. This can be viewed as the heritage of the open-source software community.

In contrast, the heritage of the commercial software community lies precisely in the challenges of development that have grown over time. Larger, more complex computer systems, communicating systems, larger-scale development, and software users with increasing expectations for functionality, performance, and ease-of-use have driven progressively more complex methods of understanding technological problems and specifying the computer programs that are intended to solve them.

Are there lessons the open-source community can take from structured requirements acquisition and specification? That the commercial software community can take from open-source requirements? These are important questions, yet they have only recently begun to be addressed by the software development and software process communities.

## 2. OPEN SOURCE, FREE SOFTWARE, AND STRANGER THINGS

One potent source of confusion in these matters is the profusion of nomenclature describing an ever-increasing number categories of computer programs. "Open Source Software", "Free Software" and "Freely-Available Software" are three of the most commonly-discussed categories of software that are distributed without cost and in source form: in line with the discussion on heritage in the previous section, this paper refers to all of these as “open-source”.

Similarly “Shareware”, “Crippleware”, “Commercial Off-The-Shelf (COTS)” and “Proprietary” software results at least to some degree from the drive to make money, either from customers or within a business. This paper will refer to all of these categories as “commercial sofware”.

There are more problematic categories, such as “Abandonware” or the infamous “Warez”. Fortunately, these are not categories that are interesting from the point of view of software development; this branch of the taxonomy can thus be ignored in the current discussion.

## 3. SOURCES OF OPEN-SOURCE REQUIREMENTS

The documents traditionally cited in discussing the motivations and methodology of open-source developers are Eric Raymond’s *The Cathedral and the Bazaar* [4] and *Homesteading the Noosphere* [5]. These writings have recently endured some relatively skeptical scrutiny. Nonetheless, they are valuable partly because they have been embraced by the open-source community, and thus illustrate some of the ideals of that community.

Perhaps Raymond’s most oft-quoted contention about open-source requirements is as follows:

> Every good work of software starts by scratching a developer’s personal itch.
>
> Perhaps this should have been obvious (it’s long been proverbial that “Necessity is the mother of invention”) but too often software developers spend their days grinding away for pay at programs they neither need nor love. But not in the Linux world—which may explain why the average quality of software originated [sic] in the Linux community is so high.

One might question the accuracy of that last sentence, inasmuch as few direct measurements of open-source software quality have been made. It is the rest of this statement, however, that deserves serious attention. It suggests the first source of open-source software requirements:

> 1. *Open-source software requirements may come from directly from the developers*

Indeed, while far from the only source of open-source requirements, this is the most obvious and common one, and also one of the sources least acknowledge (if not least practiced) in commercial software development.

The strengths and weaknesses of developer-driven requirements are reasonably clear. For example, since the developer understands their own requirements intimately, they are likely to design and implement a solution that matches them well. On the other hand, since any set of requirements collected from a single user is likely to be highly idiosyncratic, the general utility of the resulting program is likely to be lower than if a more robust requirements-gathering process is used.

Indeed, Raymond’s account of his development of the open-source program `fetchmail` in *The Cathedral and the Bazaar* lists at least two other major sources of open-source requirements: user feedback (as with any commercial product) and standards.

> 2. *Open-source software requirements may come from users of open-source software*

Certainly, as Raymond himself argues in *Homesteading the Noosphere*, a common desire of open-source developers and development organizations is to attract a large user base. This implies making a large number of users happy, presumably by providing them with software that fills their needs.

One of the largest drivers for the implementation of open-source projects is unarguably the desire to implement technical standards.

> 3. *Open-source software requirements may come from the implementation of explicit standards*

Much of the open-source software available today consists solely or largely of programs implementing pre-existing standards, from the Apache Web server to the GCC family of compilers. Paradoxically, commercial software developers often tend to ignore technical standards, either entirely or selectively, when they deem it in their best interest to do so.

The reasons for the more enthusiastic embrace of and faithful adherence to standards by the open-source community are undoubtedly many. One important reason, however, is the strong desire for interoperability in the open-source community. This may result from the greater ability of software reuse inherent in open-source, combined with the decreased ability to marshal the resources necessary for projects with large numbers of weakly-coupled requirements.

This desire for interoperability extends further, to a desire to interoperate even with programs that have no published standards or disclosed behavior.

> 4. *Open-source software requirements may come from the emulation of implicit standards*

While this is not, as Microsoft would claim, the sole source of open-source software requirements, it is certainly the case that many open-source programs have been written as functional replacements for existing commercial programs. This approach avoids problems of user interface design that are typically difficult for open-source developers, and also reduces the training and support difficulties inherent in open-source deployment. One downside is that too-literal copies of existing applications copy misfeatures and perpetuate requirements defects. Another is that ignoring other important sources of requirements may produce a product ill-suited to its target audience.

As common as the look-alike application is the interoperable one: software designed to exchange file or network data using the often proprietary undocumented formats of existing commercial applications. This sort of requirement is generally unavoidable, and is one reason why open-source developers tend to be big advocates of published interoperability standards.

Finally, open-source requirements are frequently the result of the desire to learn. Several of the authors’ closest friends are serious and well respected open-source developers. They have all at one time or another implemented some major piece of software—a programming language compiler or interpreter, a suite of UNIX utilities, a search engine—just to understand the topic.

> 5. *Open-source software requirements may come from the need to build learning prototypes*

It is often in the nature of open-source development environments and execution platforms to support this kind of rapid development of learning prototypes. Because the quality of work of the best open-source programmers is so high, these prototypes often become usable production programs.

## 4. COMPARISON AND CONTRAST: OPPORTUNITIES FOR CHANGE

Of the five sources of open-source requirements noted above, the second and third (and to some extent the first) are common sources of commercial requirements as well. The added requirements dimension of open-source might be summarized by saying that requirements in the open-source community more often come from empowered developers. This should not be surprising. Developers not motivated by the need to make a living from their software must nonetheless surely be motivated somehow. Eric Raymond's complicated and widely criticised appeal to a hypothetical gift culture might one motivation: a much more obvious motivation is that open-source developers are producing software meeting their own specific needs.

Space precludes a detailed discussion of the sources of requirements for commercial software, and much has been written about this in any case [3, 7]. Several features of commercial requirements collection and analysis seem to be largely absent in the open-source community. Absent, for example, are systematic attempts at requirements verification and validation, the use of a structured or repeatable requirements acquisition process, and the use of formal or semi-formal methods for requirements specification.

It is likely that a more traditional approach to software requirements in open-source development could pay off. None of the five sources of requirements discussed above scale well to the continued large-scale production of open-source software. As the open-source community reaches the limits of leveraging developer-driven and commercially-driven requirements, it will need to begin to think about how to construct software that more directly fills user needs. Fortunately, existing software engineering knowledge can inform such an effort in powerful ways.

There are lessons here for commercial developers as well. Approaches such as Extreme Programming [1] emphasize empowering the developer not only to gather and analyze requirements but to steer, structure and even to add to them. In teaching software engineering, one complaint that I often hear from students is that the requirements process at their organization is stultified, dysfunctional, and ultimately disconnected from the actual goals of the projects they are working on. In this situation, open-source approaches to requirements can breathe new life into the requirements process, encouraging the construction of software better suited to the needs of the developers and ultimately of the users as well.

## Acknowledgements

I have had a number of interesting discussions with Keith Packard about the issues of software engineering and open source. A recent conversation with Rick Kazman inspired the idea that the principal distinctions between open-source and commercial software are in the area of requirements. Warren Harrison has been a source of much encouragement and support in pursuing these issues.

## 5. REFERENCES

[1] K. Beck. *Extreme Programming Explained.* Addison-Wesley, 2000.

[2] F. Brooks. *The Mythical Man-Month: 20th Anniversary Edition.* Addison-Wesley, 1995.

[3] R. S. Pressman. *Software Engineering: A Practitioner's Approach.* McGraw-Hill, 2000.

[4] E. S. Raymond. The cathedral & the bazaar. Web document, URL `http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/` accessed Fri Mar 15 20:39 UTC 2002. Reprinted in [6].

[5] E. S. Raymond. Homesteading the noosphere. Web document, URL `http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-homestea%ding/` accessed Fri Mar 15 20:39 UTC 2002. Reprinted in [6].

[6] E. S. Raymond. *The Cathedral & the Bazaar.* O'Reilly, 2001.

[7] I. Sommerville. *Software Engineering.* Addison-Wesley, 2000.