

# ULRR

## A qualitative study of open source software development: the OpenEMR project

Item Type	Meetings and Proceedings
Authors	Noll, John;Beecham, Sarah;Seichter, Dominik
Citation	5th International Symposium of Empirical Software Engineering and Measurement;2011
Publisher	IEEE Computer Society
Download date	2026-05-09 14:39:32
Item License	<a href="https://creativecommons.org/licenses/by-nc-sa/1.0/">https://creativecommons.org/licenses/by-nc-sa/1.0/</a>
Link to Item	<a href="https://hdl.handle.net/10344/2060">https://hdl.handle.net/10344/2060</a>

## A Qualitative Study of Open Source Software Development: the OpenEMR Project

John Noll, Sarah Beecham, and Dominik Seichter

*Lero - the Irish Software Engineering Centre*

*Department of Computer Science and Information Systems*

*University of Limerick, Limerick, Ireland*

*e-mail: {john.noll,sarah.beecham,dominik.seichter}@lero.ie*

**Abstract**—Open Source software is competing successfully in many areas. The commercial sector is recognizing the benefits offered by Open Source development methods that lead to high quality software. Can these benefits be realized in specialized domains where expertise is rare? This study examined discussion forums of an Open Source project in a particular specialized application domain – electronic medical records – to see how development roles are carried out, and by whom. We found through a qualitative analysis that the core developers in this system include doctors and clinicians who also use the product. We also found that the size of the community associated with the project is an order of magnitude smaller than predicted, yet still maintains a high degree of responsiveness to issues raised by users. The implication is that a few experts and a small core of dedicated programmers can achieve success using an Open Source approach in a specialized domain.

**Keywords**—Open Source Software; Electronic Medical Records; Qualitative Research; Inter-rater Reliability; Cohen's kappa

### I. INTRODUCTION

There is increasing interest in applying the Open Source approach in domains that are not traditional Open Source territory. For example, the U.S. Department of Defense has established an Open Source “forge” in an effort to promote sharing of software among DOD contractors [1], [2]. Similarly, the European Space Agency has requested tenders for a feasibility demonstration of an Open Source repository of source code to support space missions [3].

These domains would seem like a poor fit for the Open Source approach, which on the surface is a much less formal way of developing software [4]. Also, several studies have suggested that users play a central role in the development of Open Source software, as both programmers and contributors of requirements [5]; one would not expect many soldiers or astronauts to be programmers. Further, it has been hypothesized that a successful Open Source project requires an active community of developers and testers that is much larger (two orders of magnitude) than the “core” developer group [6]. If this is the case, there may not be sufficient participants in specialized domains such as space software to support successful Open Source software development [1].

The evidence supporting these assertions is largely drawn from case studies of well-known, successful Open Source

projects such as the Apache Web Server [6], Mozilla Web Browser [7], [8], and Linux operating system [9], [10]. These projects exist in the traditional strongholds of Open Source software: Internet infrastructure and programming tools [5], where one would expect the majority of developers to also be users of the product they are developing. The Apache Web Server, for example, began as a loose collaboration among web site administrators who contributed code (“patches”) to fix bugs or add features to the original NCSA web server [11].

This study attempts to answer the question,

*Can the Open Source approach work in specialized domains such as aerospace and defense software development, where the users (pilots, astronauts, soldiers, etc.) would not be expected to participate directly in the development of the software?*

This paper presents a case study of the OpenEMR project ([www.oemr.org](http://www.oemr.org)), which produces an Electronic Medical Record (EMR) system for use by physicians, clinic administrators, and other medical practitioners to manage their practices. Electronic medical records maintain a history for patient encounters as well as a record of patient background, diagnoses, prescriptions, and billing, and are tightly integrated into medical practice workflow.

This case is interesting because the OpenEMR project develops software for a highly specialized domain, where one would not expect the users of the system – doctors, nurses, administrators, clerks – to be programmers as well. Further, the requirements for such products are determined by a variety of factors, including regulations specifying patient confidentiality [12], and standards for electronic billing (see, for example, [www.x12.org](http://www.x12.org)). As such, the processes employed by such a project would be expected to differ from those followed by projects in conventional Open Source strongholds.

A single case cannot be expected to provide results that would be generalizable to a majority of situations. Rather, a case study can reveal interesting characteristics of a problem that can provide the conceptual foundation for broader methods such as surveys or controlled experiments. Also, as more case studies of a particular phenomenon are performed, a general view begins to emerge through meta-analysis [13]. Finally, a single case can serve as a

counterexample, providing *falsifiability* of a theory [14].

This latter characteristic motivates the current study. By examining a project in a domain where users would not be expected to be programmers, it challenges the prevailing assumption that a successful Open Source project requires a significant number of users among its core developers [7]. And yet, as will be shown in Section IV, there are doctors writing code for OpenEMR.

This study employed *content analysis* [15] to classify a total of 1218 OpenEMR discussion forum messages. The analysis revealed that core developers contributed nearly half of all messages, almost three-quarters of implementation announcements, and over half of bug fixes and proposals for enhancements.

These results are significant because they show, in contrast to earlier hypotheses [7], that open source projects can be successful with a relatively small community.

The remainder of this paper is organized as follows: the next section describes the OpenEMR project and the case study; following that is a detailed description of the method, including use of inter-rater agreement analysis to refine the coding scheme, used to analyze the project. Section IV presents the results of the study, followed by related work in Section V, and conclusions in Section VI.

## II. CASE STUDY

As mentioned in Section I, this paper presents results of a case study of a small Open Source project in a specialized application domain. The project, OpenEMR, develops an electronic medical record (EMR) product for small medical practices and clinics. OpenEMR began as a conventional commercial development effort called “MP Pro” [16]. The first version of MP Pro was released in 2001. In 2002, the name was changed to “OpenEMR” and the code released as Open Source under the GNU General Public License [17]. In 2005, the source code, issue database, and discussion forums were transferred to *Sourceforge.net*; this marks the beginning of the project’s history examined by this case study, which examines data from 2005 to 2010.

### A. Research Questions

The study attempts to answer five research questions inspired by Mockus and colleagues’ seminal case study of the Apache Web Server project [6]. As part of that study they proposed a set of hypotheses about successful Open Source software projects, including:

- Successful projects will have a core of no more than 15 developers.
- A large group of developers surrounding the core will repair defects.
- Projects without a large group of developers repairing defects will fail due to poor code quality.
- Developers of successful Open Source products will also be users of the product.

- Open source projects will respond rapidly to customer problems.

One would expect a product like Apache to have many users among its developer population. But would this also hold in a narrow domain with a highly specialized user community, where the available population of users who might be interested in contributing, and have the necessary skills to contribute to an Open Source project, would be limited? If the Core Developers must include a significant number of end-users, and end-users with sufficient skill to be developers are rare, then a domain with these characteristics may only be able to support a few Open Source projects.

This leads to the following research questions:

- RQ1:** Is the OpenEMR project successful?
- RQ2:** What is the size of the core developer group?
- RQ3:** Who fixes bugs in OpenEMR?
- RQ4:** How many of the OpenEMR Core Developers are also users of the product?
- RQ5:** How long does it take OpenEMR project members to respond to issues?

### B. Study Approach

OpenEMR makes extensive use of the discussion forum features of *Sourceforge.net*: requirements, development issues, and new implementations are announced and discussed in the developers’ forum, while bugs, requests for enhancement and other issues are posted to the developers’ forum, users’ forum, and the help forum. Although *Sourceforge.net* provides an “issue tracker” (bug database), as well as a “feature tracker” for recording requirements, until recently OpenEMR has not used these forums. There is also a “patch tracker” and “code review tracker” that have been used sporadically to review submissions of new code from non-core developers.

Given that the discussion forums are the primary means for communicating and recording important activity in the OpenEMR project, content analysis is an appropriate method for answering the research questions; this method is described in the next section.

## III. METHOD

A great variety of software project metrics can be extracted automatically from project repositories, and web sites aggregating data on open source projects, such as Ohloh ([www.ohloh.org](http://www.ohloh.org)), FLOSSmole ([flossmole.org](http://flossmole.org)), and FLOSShub ([flosshub.org](http://flosshub.org)), continually collect data on tens of thousands of open source software projects, computing metrics on number and size of modules and commits, number of developers, downloads, users, bug reports, etc.

But some kinds of software project data - bug reports, commit log entries, email messages, and forum postings - include blocks of natural language text where meanings are difficult to analyze automatically. As software development is a human-intensive activity, these qualitative data

convey important information about a project that cannot be explained by numbers alone. For example, analyzing project discussion forum postings can help to explain who is reporting and fixing bugs, who actually commits enhancements, how these people are employed, how requirements are elicited, and how users are supported.

While qualitative techniques employing human interpretation are necessary to analyze such data, qualitative analysis is a labor-intensive activity; as such, the amount of data that can be analyzed is limited by the capacity of human researchers.

This study employed a qualitative mining technique that is transparent and repeatable, leads to objective findings when dealing with qualitative data, and is efficient enough to be applied to large archives. In particular, inter-rater agreement analysis was used to develop a classification scheme to categorize natural language statements such as discussion forum posts; the scheme is both accurate and efficient enough to analyze thousands of such statements.

The method used in this study is based on *content analysis*. Content analysis is a classification technique that is often used to analyze interviews and focus group data. The process of data analysis as described by Krippendorff [15] is similar to the grounded theory method, where replicable and valid inferences are made from the data to their context. Where content analysis differs from grounded theory is that it is largely numeric and therefore includes a quantitative form of research. Content analysis produces results such as, “13% of messages posted to discussion forums expressed proposals for new functionality.”

Content analysis involves assigning a type or *code* to excerpts of text [15], which captures or classifies the meaning of the excerpt. Traditionally, qualitative analysis is performed on large documents such as interview transcripts; individual sentences or fragments within the larger document are coded according to their meaning or intent. In the context of software repository mining, however, the “documents” are often short notes such as commit log entries, bug reports and associated comments, and forum postings. Consequently, each artifact can be considered as a whole, with a type assigned to convey the meaning of the entire artifact.

For example, some issue tracking systems do not explicitly distinguish between failure reports and requests for enhancements. Using content analysis, a researcher can assign a type code to each report, classifying it as a bug, request for enhancement, duplicate, etc.; the resulting coded sample set gives a picture of how the issue tracking system might be used to capture new requirements as well as product failures.

#### A. Developing a Coding Scheme

Content analysis aims to identify the meaning of text by assigning a code that conveys that meaning. Coding allows researchers to ask quantitative questions about qualitative data, such as, how many times do core developers post

Table I  
INITIAL MESSAGE CODES

Code	Meaning
answer	Response to an issue, query, or request.
asserted-feature	Assertion, by developer, of the need for a new feature.
asserted-enhancement	Assertion of the need for an enhancement to a feature.
bug-fix	Announcement of code to fix a bug.
bug-report	Report of failure in code.
comment	Comment on an assertion, bug-report, or proposal.
documentation	Announcement of new end-user documentation.
fix	Same as bug-fix.
gossip	Some light comment about the project or broader EMR domain.
implementation	Announcement of new code implementing a feature or enhancement.
issue	Message describing some problem encountered using or installing the product.
minutes	Minutes of conference calls.
news	Announcement of some news that might affect the project, such as start of a rival project or new regulation.
org	Message about the organization or governance of the project.
process	Statement or suggestion about project development processes.
proposed-feature	Proposal for a new feature.
proposed-enhancement	Proposal for enhancement of existing feature.
query	Question about some aspect of product, such as what should be expected behavior.
request-for-clarification	Response to a bug report message, requesting more information.
clarification	Response to request for clarification.
request-for-information	Request, usually by developer, for information about how a feature works or how to modify it.
information	Response to request-for-information.
rfe	Request for Enhancement, usually by end user.
other	Not one of the above.

assertions of requirements to discussion forums? As such, it is essential that the coding scheme used to convey meaning is accurate. Also, it must be repeatable: different researchers should assign the same code to a given text fragment, and the same researcher should assign the same code to a given fragment when analyzed a week or a month later.

But a good coding scheme is not only accurate and repeatable; if the number of codes is small, and their definitions are clear, the coding process becomes straightforward and can be completed easily and quickly.

Our coding method is adapted from Burnard [18] and comprises the following steps:

1) *Create Initial Type Set*: The first step is to select a representative sample of text fragments, and from these, create an initial set of codes that capture their meanings. This is an inductive approach, in which the researcher reads a fragment and invents a code (word or phrase) that captures the meaning conveyed by the fragment. The list of codes grows and evolves as more fragments are read, and in the end may have many codes.

Table II  
FINAL MESSAGE TYPES AND CODES.

Type	Code	Meaning
Implementation	<i>impl</i>	A post announcing code implementing new functionality, or enhancing existing functionality.
Bug Fix	<i>fix</i>	An announcement of code to fix a bug, or a bug fix patch submitted to the bug fix or code review tracker.
Proposed Change	<i>prop</i>	A description of a proposed enhancement or new feature, or request for enhancement or new functionality.
Issue	<i>issue</i>	A bug report, or request for help with some issue involving configuring or running the product.
Other	<i>other</i>	None of the above.

An initial set of codes was derived by the first author from a trial examination of several hundred messages from the three discussion forums (developers, users, and help) and three “trackers” (bugs, patches, and code review) for OpenEMR on *Sourceforge.net*. This initial set, shown in Table I, attempted to capture the wide variety of message meanings, and so comprised a total of 24 type codes.

2) *Aggregate into Type Categories*: Next, the list of codes is examined to discover broader categories. Types with similar meaning are grouped together, and coalesced into a single category. The goal is to refine the list into a handful of categories with distinct meanings, so that it is easy to decide to which category a given text fragment belongs. The categories are given names which become the codes that are assigned to text fragments.

The initial set of 24 types was refined to five categories, shown in Table II, that capture meaning appropriate to the research questions for this study. For example, the initial types *issue*, *query*, *bug report*, *request-for-clarification*, and *request-for-information* were combined into a single category called ‘Issue.’

The catch-all category called “other” deserves some explanation. The OpenEMR discussion forums include a wide variety of messages, concerning project governance, news and trends in the larger EMR domain, and gossip about project members or even competitors: the initial type list shown in Figure I gives some idea of this diversity.

However, the focus of this study is on software development activities: who proposes and implements new features, who reports and fixes bugs, how fast does the community respond to issues, etc.; the “other” category was created for messages that do not specifically concern software development, even though these messages comprise nearly half of the sample.

3) *Create Checklist*: Once a disjoint set of categories is created, a checklist describing how to categorize a given text fragment is developed. This is an important tool for the coding process, as it provides a step-by-step decision process

Table III  
CROSSTABULATION TABLE COMPARING CODING OF TWO RESEARCHERS.

John	Sarah					Total
	<i>fix</i>	<i>impl</i>	<i>issue</i>	<i>other</i>	<i>prop</i>	
<i>fix</i>	3	0	0	3	0	6
<i>impl</i>	0	2	0	2	0	4
<i>issue</i>	1	1	15	0	0	17
<i>other</i>	1	1	2	21	2	27
<i>prop</i>	0	0	0	2	3	5
Total	5	4	17	28	5	59

for the researcher to use to code the data. The final checklist used for this study is included in the Appendix.

4) *Refine Checklist*: The set of codes and associated checklist are evaluated and refined using a series of trials involving two or more researchers. The goal is to achieve a high degree of agreement among researchers about which code should be assigned to a text fragment. A small sample of fragments is coded independently by two researchers, and the results compared to see how well they agree. Disagreements are discussed to determine how the checklist or set of codes could be refined to make the choice of correct code more clear. The process is repeated until an acceptable level of agreement is achieved.

For this study, a trial set of 94 messages was coded by both the first and second authors, and the results compared using crosstabulation.

A crosstabulation comparing code assignments of two researchers shows exactly how two researchers assigned each code, and thus which codes are most ambiguous or problematic. In the example shown in Table III, each cell in the table shows the number of messages coded with the row label by the first researcher (John) that were coded with the column label by the second researcher (Sarah). The diagonal, therefore, represents agreement. For example, in Table III, three out of five of Sarah’s *fix* assignments agreed with John’s, while three out of six of John’s *fix* assignments agreed with Sarah’s.

The message types and definitions were refined with the aid of the crosstabulation comparing how each author coded the sample set. The disagreements were discussed, and revisions made to the checklist. The trial coding was then repeated on new samples of between 42 and 136 messages. This trial revealed continuing problems with the definitions of *impl*, *prop*, and *issue* categories.

5) *Assess Inter-rater Agreement*: Agreement between different researchers about which codes to assign is termed inter-rater agreement, inter-rater reliability, or sometimes inter-coder reliability, and is an indication of the reliability and repeatability of a coding scheme.

There are several statistics that can be used to assess the degree of inter-rater agreement; for this study, Cohen’s kappa

Table IV  
BENCHMARKS FOR MAPPING COHEN’S KAPPA TO LEVEL OF AGREEMENT [21].

K	Strength of agreement
0	poor
0.01–0.20	slight
0.21–0.40	fair
0.41–0.60	moderate
0.61–0.80	good
0.81–1.00	almost perfect

was chosen to assess overall inter-rater agreement due to its wide use [19], [20], as well as support in common statistical tools. Cohen’s kappa is a statistic that attempts to assess the degree of agreement between the codes assigned by two researchers working independently on the same sample. Cohen’s kappa takes into account the fact that a certain level of agreement would be achieved even if codes were assigned randomly; thus, it is more conservative than percentage agreement, which does not account for random agreement.

The kappa statistics for the seven trials used to develop and refine the checklist for this study, as well as the statistic for the main coding exercise, are shown in Table V and Figure 1.

Meaningful values of Cohen’s kappa fall between 0 and 1, where 0 indicates poor agreement, and 1 perfect agreement. Landis and Koch [21] proposed a set of thresholds for mapping kappa values to strengths of agreement; these are shown in Table IV.

### B. Coding the Data

This phase involves several researchers and a large sample. Our experience shows that three researchers working part-time with a well-refined coding scheme can easily code over 1000 discussion forum entries in a week.

Messages posted to *Sourceforge.net* discussion and tracker forums are grouped by ‘topic’ (sometimes called a ‘thread’); the topic is established by the first message, and provides a focus for the content of subsequent messages. To reduce the impact on *Sourceforge.net*’s resources, the messages were downloaded in small chunks of twenty-five topics and related messages over a period of four weeks in 2010. A script extracted the subject of each topic, along with the author’s ID, date, and body for each message related to the topic, and placed these data in a file formatted to facilitate coding.

A large sample of the entire collection of messages was coded. We selected, at random, 181 topics from the 2676 total topics, comprising 1218 messages of the 14917 total messages contained in all topics in all forums. The coding effort was divided equally among all three authors, by allocating a third of the topics to each. As a final check of agreement, a subset of 367 messages was coded by both the first and second authors; Cohen’s kappa was computed

Table V  
COHEN’S KAPPA STATISTICS FOR SUCCESSIVE REFINEMENT ITERATIONS.

iteration	1	2	3	4	5	6	7	8
# msgs	94	42	136	54	59	118	40	367
kappa	.46	.63	.52	.65	.63	.55	.80	.65

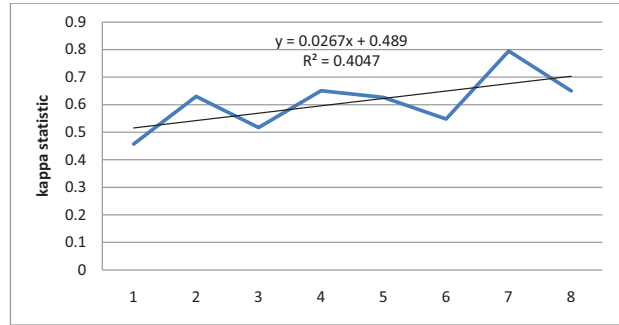


Figure 1. Kappa statistics for successive refinement iterations.

for this subset, yielding a value of .65 indicating “good” agreement.

Quantitative analysis was performed on the resulting coded sample, as described in the next section.

### C. Identifying Participants

The frequency and type of messages posted by contributors provide insight into the size and makeup of the OpenEMR community. Participants in OpenEMR discussion forums are required to register before they can post messages to a topic or submit a new topic. As part of the registration process, participants choose both a “username” used by the system for authentication, and a “displayed name” for identifying the author of a message to human readers; both are included with each message. Once the message sample was coded using content analysis, a script was created to extract the code, date, and author of each message and accumulate these statistics; these are shown in Table VI.

## IV. RESULTS

As mentioned in the previous section, both automated data extraction and content analysis were performed on the downloaded corpus. The results are presented below.

### A. Is OpenEMR successful?

One indication of project success is the amount of activity associated with the project, which can be measured by code committed to the source code repository, messages posted to the project discussion forums, and the number of different people involved in these activities [22].

OpenEMR is a small project that nevertheless has an enthusiastic community of developers, contributors, and users, and a significant installed base [23]. Figure 2 depicts three measures of community activity associated with OpenEMR.

Table VI  
DISTRIBUTION OF MESSAGE TYPES IN STUDY SAMPLE

Group	impl	fix	prop	issue	other	Total
Core Dev.	49	39	79	70	354	591
% of type <sup>a</sup>	72%	59%	51%	21%	60%	49%
% of Group <sup>b</sup>	8%	7%	13%	12%	60%	100%
Others	19	27	76	267	238	627
% of type	28%	41%	49%	79%	40%	51%
% of Group	3%	4%	12%	43%	38%	100%
Total	68	66	155	337	592	1218
% of Total <sup>c</sup>	6%	5%	13%	28%	49%	100%

<sup>a</sup> Fraction of messages, of type identified by the column heading, that are submitted by this Group. Thus, the Core Dev. Group submitted  $\frac{49}{68}$  or 72% of the 68 *impl* messages, while  $\frac{19}{68}$  or 28% of *impl* messages were submitted by Others.

<sup>b</sup> Fraction of all messages posted by this Group that have type identified by the column heading. So,  $\frac{49}{591}$  or 8% of the 591 messages submitted by the Core Dev. Group were of type *impl*, while  $\frac{19}{627}$  or 3% of messages submitted by Others were of type *impl*.

<sup>c</sup> Fraction of all messages examined having type identified by the column heading. Thus,  $\frac{68}{1218}$  or 6% of all messages were of type *impl*.

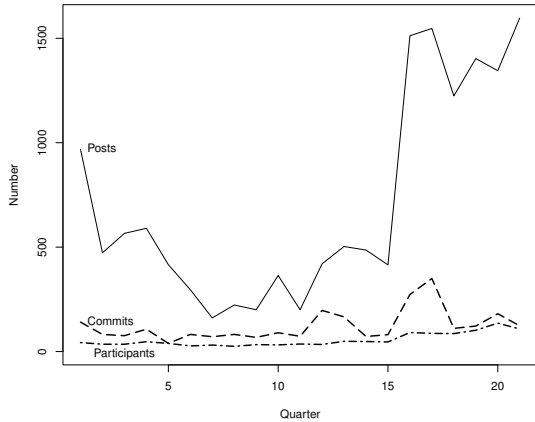


Figure 2. OpenEMR growth, as measured by number of forum posts, code commits, and active participants per quarter.

The first measure (labelled ‘Posts’) is the number of messages per quarter posted to all OpenEMR discussion forums on *Sourceforge.net*. After an initial spike of nearly 1000 messages/quarter when the development of OpenEMR was moved from the Pennington firm to *Sourceforge.net*, there was a decline in activity for the next two years to below 200 messages/quarter; this has been followed by a period of increasing activity, with a significant increase to nearly 1500 messages/quarter beginning about eighteen months from the end of the study period; this rate has been maintained to the end.

Commits to the OpenEMR source code repository (labelled ‘Commits’ on Figure 2) are somewhat volatile, with a slight initial decline followed by steady activity of about 100 commits/quarter punctuated by several spikes at three and four years into the study period.

The number of different people observed participating in discussions in the OpenEMR forums in a given quarter (‘Participants’ on Figure 2) has increased slowly but steadily from 35/quarter at the beginning of the study period to more than 100/quarter at the end.

These measures suggest that OpenEMR is a successful, active project with a small but growing user community.

### B. Who are the core developers?

The discussion forums for OpenEMR contain 613 unique user ids. Since users must register to participate in forums or commit changes to the repository, this number gives a good picture of the size of the user and developer population.

The OpenEMR community is quick to grant commit access to new developers: once an individual submits a few bug fixes or enhancements that meet with the approval of the existing Core Developers, that individual will be offered commit access so he or she can submit code directly to the code repository. So, an enthusiastic developer will quickly move from Bug Fixer to Core Developer.

Potential Core Developer IDs were extracted from the CVS commit logs. The intersection of these IDs with the forum author IDs revealed that 16 of 19 commit log IDs also appeared in the discussion forums; these were deemed to comprise the Core Developers. Two of the three remaining CVS IDs appear to be associated with programmers employed by some of the Core Developers; the third of these remaining CVS IDs, found in a single commit log entry, appears to be that of a *Sourceforge.net* admin account.

### C. Who fixes bugs in OpenEMR?

Mockus and colleagues hypothesize that a successful project will have a group of developers outside the core that fixes most bugs, leaving the core to focus on enhancing the product; projects that do not have such a group will not be successful, as the Core Developers will be preoccupied with bug-fixing and won’t be able to implement new features [6]. However, Table VI shows that, despite being only a small part of the OpenEMR community, the Core Developers contributed almost 60% of bug fixes in the sample, in addition to nearly three-quarters of the new features or enhancements. This indicates that the Core Developers can perform both roles successfully.

### D. How many of the OpenEMR Core Developers are also (end-) users?

In addition to the technical content related to OpenEMR and its development effort, the OpenEMR forums contain a great deal of information about the background and interests of the community members. It is not uncommon for a new developer to submit an introductory message to the developer forum, as a way of discovering what features or open issues might match his or her skills and interests. Also, developers will introduce background information about

Table VII  
OCCUPATIONS OF CORE DEVELOPERS

Group	Num.	Frac. of Total
Consultant	8	50%
Programmer employed by clinic	3	19%
Physician/Health Practitioner	3	19%
Clinic IT manager	2	12%
Total	16	100%

Table VIII  
TIME TO RESPOND TO ISSUES

Number of Issues		Time to Resolve			
Reported	Resolved	Mean	Median	Min.	Max.
159	151	11.74hr <sup>a</sup>	2.3 hr	2.2min	124hr <sup>b</sup>

<sup>a</sup> Excluding 3 outliers and 8 non-responses.

<sup>b</sup> Excluding 8 non-responses.

the environment in which they work, to lend credibility to assertions about requirements or design, or to provide context for explaining issues.

As such, by searching the OpenEMR *Sourceforge.net* forum corpus for posts associated with the Core Developer user IDs, the professional occupation of most Core Developers could be discovered, as shown in Table VII.

Of most interest are the three Core Developers who are physicians. One of these hosts the OpenEMR web site, established and chairs the not-for-profit corporation associated with OpenEMR, and provides much of the end-user support on the OpenEMR forums. Another is a Swedish doctor who developed the OpenEMR language internationalization facility. The third is a medical doctor who with a strong interest in programming.

The three physicians are true end-users of medical record software; the IT administrators could also be considered end-users, or at least consumers, as they would be deploying OpenEMR in their own settings. Considering that the seven consultants would have significant understanding of end-user needs by virtue of their close interaction with clients who are end-users, the majority of OpenEMR Core Developers (over 80%) have deep domain knowledge that they can bring to bear in enhancing and extending OpenEMR's functionality.

#### E. How long does it take to respond to issues?

Response to issues was calculated by examining the messages of type *issue* in the code sample. For each issue identified in the sample, an attempt was made to identify a post that represented a response to the issue or query raised, by looking forward in time for a post of type *other*. The response could be an answer to the query posed, a suggested solution or workaround for a problem, an acknowledgement that the author had indeed found a bug in the system, or simply a request for more information about the problem. The difference between the times when the *issue* and response

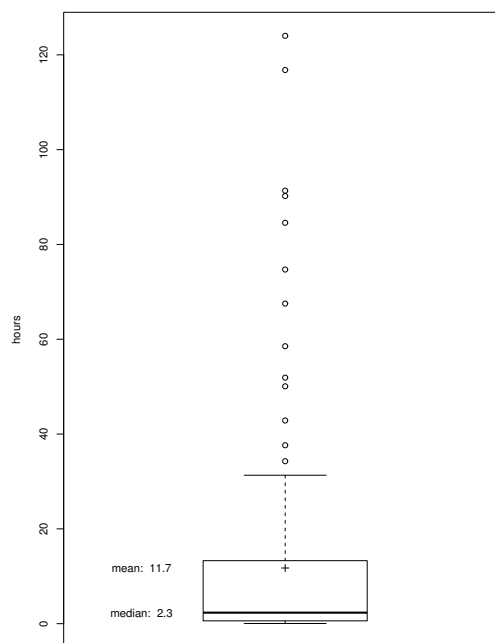


Figure 3. Time to respond to issues.

were posted was calculated using the time-stamps supplied by *Sourceforge.net*.

Because of its small size, we were surprised that the OpenEMR community is so quick to respond to questions and other issues posted by users (Figure 3). Table VIII shows that, while response time varies widely, many queries receive a response within a few hours, with a median of little more than two hours, and some within minutes.

#### F. Summary

To summarize, the research questions proposed in Section II were answered as follows:

**RQ1:** Is the OpenEMR project successful?

Increasing project activity over time, as measured by discussion activity, code commits, and number of participants, supports the conclusion that OpenEMR is successful.

**RQ2:** What is the size of the core developer group?

Analysis of discussion forums and commit logs shows 16 core developers. This is consistent with Mockus and colleague's study [6].

**RQ3:** Who fixes bugs in OpenEMR?

In contrast to the hypothesis of Mockus and colleagues, our analysis shows that the majority of bugs are fixed by core developers. This indicates that a small core of dedicated programmers can both maintain and enhance an open source product.

**RQ4:** How many of the OpenEMR Core Developers are also users of the product?

Three core contributors are health care practitioners, and two more are clinic administrators, meaning over 30% of Core Developers are end-users. This tends to support

Mockus and colleagues’ hypothesis that at least some of the Core Developers of a successful open source project must also be end-users.

**RQ5:** How long does it take OpenEMR project members to respond to issues?

Response to issue inquiries posted to the discussion forums is excellent, with a median response of 2.3 hours.

### G. Limitations

Any empirical study has limits on the validity of the results and the extent to which they apply in other contexts. The following threats to validity have been identified.

1) *Threats to Construct Validity:* Construct validity is the extent to which the measures employed to characterize a real-world phenomenon (‘construct’) actually measure that phenomenon. This applies to the case study, rather than the method itself. The chief threats to construct validity are the definition of Core Developer, and the choice of codes, especially the *issue* category.

There are some active contributors to OpenEMR who write documentation and provide expertise, but do not commit code to the repository; these individuals would not fall under our definition of Core Developer. However, from the standpoint of evaluating our research questions, this is not an issue, as we focus largely on *programming*: developing new features and fixing bugs.

The definition of the *issue* category includes far more than verified bugs, and as such is broad. As a consequence, our results overestimate the contributions of users outside the core: many issues appearing in the discussion forums are the result of misunderstandings about installing and configuring OpenEMR, rather than actual failures of the code.

2) *Threats to Internal Validity:* Internal validity refers to how accurately the research method extracts the relevant measures. This is most relevant to our choice of inter-rater reliability statistic. Cohen’s kappa, while popular, has been criticized by some researchers due to the way it attempts to account for randomness: these researchers point out that, when faced with an uncertain choice between two codes, coders do not choose one over the other randomly, but rather apply some decision-making rationale that is not captured by Cohen’s kappa. Nevertheless, as discussed in Section V Cohen’s kappa is widely used and as such is a reasonable choice.

Our sampling method might also affect internal validity. In order to preserve the context around each message, we selected a random sample of discussion *topics*, rather than individual messages. While the resulting message sample is large, there might be some topics that certain participants are not interested in, and therefore they may not have participated in the topics in our sample. However, 181 topics (comprising 1218 messages) covers a wide range of subjects, so we are confident that the results represent the actual range of activity in the OpenEMR community.

Our sampling method does not allow us to accurately characterise the relative size of the Core Developers, Bug Fixers, and Bug Reporters. This is because we sampled *topics*; we would need to examine messages from a random sample of *users* to determine the relative size of each Group. Nevertheless, we are able to determine the size of the *contribution* made by the Core Developers, which is one of the important results of this study.

3) *Threats to External Validity:* External validity addresses the extent to which a study’s results apply to the world in general.

A single case study does not establish a general trend; at best it can falsify a hypothesis, showing where the hypothesis needs to be revised. This is the case with our case study results, which show that the Core Developers do most of development work on a small open source project, including fixing bugs.

Although not the main focus of this work, our coding scheme and checklist could be applied to any repository of messages or other text artifacts comprising discussions about development issues related to a software project. This would include issue databases as well as discussion forums and mailing lists. More importantly, our approach to analyzing such repositories using qualitative content analysis is applicable to any repository containing discrete blocks of text.

## V. RELATED WORK

Testing inter-rater reliability is not a new concept, and has been used in software engineering qualitative research extensively over the years. The aim of inter-rater reliability tests is to gain an external understanding of the level of agreement between how two independent researchers classify qualitative data that is open to subjective interpretation.

For example, in a process improvement initiative performed by Henningsson and Wohlin [24], a group of raters classified faults. These researchers used Cohen’s kappa statistic to measure whether 8 people could agree on how to classify faults independently. Other researchers used the same method to test the reliability of their fault classifications, e.g. El Emam and Wiczorek [25] and Hall et al [26], with mixed results. Henningsson and Wohlin [24], and Hall et al [26], were unable to obtain a good interrater agreement. In both these cases, the interrater measure was used to test agreement post hoc rather than to feed back where problems and differences of opinion occur. However El Emam and Wiczorek [25], in testing the repeatability of code defect classifications report some of the benefits of using kappa, for example when their tests returned a poor kappa coefficient they used this to go back to look at specific fault types that were causing problems for the researchers.

Software Process Assessment is another area in Software Engineering that has used Cohen’s kappa statistic to test the external reliability of interrater agreement [27], [28], [29],

[30], [31]. As software process assessment can be subjective, researchers identify the need to check the reliability of the results. This research area appears to be more mature in the use of applying statistical techniques to look at agreement levels.

Some researchers have identified that the kappa coefficient is not always an appropriate measure of agreement due to what is known as the ‘Kappa Paradox’ [31], where in certain circumstances high agreement is accompanied by a low kappa statistic. However, El Emam and colleagues did find that ensuring high interrater agreement could lead to a reduction in the cost of assessments that include organizational processes in their scope [29], and make a good case for using the kappa coefficient [25].

Other areas in Software Engineering that have used this method include Fuzzy Systems [32], and in the subjective evolvability evaluation of object-oriented software [33]. Also, Vilbergsdttir et al [34] used kappa statistics over several iterations to revise their scheme defining usability attribute values.

We argue that as this method is used in critical health areas [25], it should have the required rigor to be used in testing software engineering classification agreement terms.

## VI. CONCLUSIONS

This paper presented a case study of a small open source project developing software for a specialized domain. The study makes two main contributions.

First, the results of the case study of OpenEMR show that a small community of enthusiastic programmers and users can sustain an open source software project over the long term. This is significant, as it shows that the open source approach can succeed in specialized domains where one would expect to find relatively few qualified programmers among the user community.

Second, the application of inter-rater agreement analysis in the case study demonstrates how an effective coding scheme can be created that is transparent, repeatable, and consistent. This allows several researchers to work independently on content analysis, while still producing results that can be reliably aggregated.

## APPENDIX

Following is the final coding checklist used to guide researchers during coding of messages.

### Coding Checklist

- 1) Is the post a *verbatim* repeat or duplicate (as in, no new information whatsoever) of a previous post *in the same topic*? If ‘yes’ type=*other*.
- 2) Does the post mention a commit, patch, code, ‘form,’ installer, tar archive (‘tarball’), end user documentation, or change to the OpenEMR web site? (Forms are how end-users interact with OpenEMR; they are specified in a form specification language and may require some backend PHP code.)
  - a) Is the code complete and available for use, as a download or in the source code repository?
    - i) Does the post announce a *new* enhancement or functionality?
      - A) Does the post refer to a bug, either previously reported or documented in the post itself? If yes, type = *fix*;
      - B) Otherwise, type = *impl*
    - ii) Otherwise, if the poster is just referring to something previously implemented (to, for example, answer someone’s question about how to do something), type = *other*
- 3) Does the post include the phrase “I’m going to (implement, finish, write, create) ...,” “I’m working on ...” or “I’ve started ...” associated with a feature, enhancement, installer, tar file, or document? If yes, type = *prop*;
- 4) Does the post ask for opinions on a new feature, enhancement, installer, tar file, or document? Does the post include phrases like “is there any interest in ...” referring to some not yet existing feature or enhancement or documentation?
  - a) Does the post ask for opinions on *working* code or new “forms” that the poster is making available, via email or download? If yes, type = *impl*;
  - b) Does the post ask for opinions on new functionality, forms, or documentation that the poster intends to implement? If yes, type = *prop*;
  - c) Otherwise, type = *other*.
- 5) Does the post comment on a previous post that suggests an enhancement or new functionality?
  - a) Does the suggest how the proposed enhancement or new functionality might be implemented?
    - If yes, type = *prop*;
    - b) Does the post affirm the proposal, but offer suggestions changes to the proposed functionality? If yes, type = *prop*;
    - c) Does the post offer an alternative to the proposal? If yes, type = *prop*;
    - d) Otherwise, type = *other*.
- 6) Does the post include phrases like “it would be nice to have ...” or “I would like to see ...” or “it would be useful to have ...” referring to some not yet existing feature or enhancement or documentation? If yes, type = *prop*;
- 7) Does the post ask whether certain functionality exists?
  - a) Does the poster state or imply that if it doesn’t, he will implement it? If yes, type = *prop*;
  - b) Does the poster state or imply that if it doesn’t, he would like someone to implement it? If yes, type = *prop*;
  - c) Otherwise, type = *issue*.
- 8) Does the post:
  - a) mention an error or problem the poster is having with OpenEMR, or some unexpected behavior of, or output from, OpenEMR, or mention difficulty getting OpenEMR to do something?
  - b) ask whether certain observed behavior of, or output from, OpenEMR is correct or expected?
  - c) ask how to get OpenEMR to do something, or how to modify specific code to enhance or implement a feature, or how to configure OpenEMR to do something or run on a specific platform?
  - d) provide additional information about an issue raised in a previous post, or follow up an issue raised in a previous post?
  - e) describe how some suggested fix or intervention did not work?
  - f) describe how some previously suggested fix worked, but now there is a new issue, problem, or difficulty?
  - g) ask a general question about OpenEMR?
- 9) If the answer is ‘yes’ to one of the above, type=*issue*.
- 9) If the answer is ‘no’ to all of the above, type=*other*.

## ACKNOWLEDGMENTS

This work was supported, in part, by Science Foundation Ireland grant 03/CE2/I303\_1 to Lero - the Irish Software Engineering Research Centre ([www.lero.ie](http://www.lero.ie)).

## REFERENCES

- [1] S. Hissam, C. B. Weinstock, and L. Bass, “On open and collaborative software development in the DoD,” in *Proceedings of the Seventh Annual Acquisition Research Symposium*.

- Naval Postgraduate School Acquisition Research Program, May 2010.
- [2] U.S. Department of Defense, Defense Information Systems Agency, "Forge.mil," Web site, accessed 15 August, 2010. [Online]. Available: [www.forge.mil](http://www.forge.mil)
  - [3] "Feasibility of open source software repository," European Space Agency Invitation to Tender AO/1-6301/09/NL/AF, Nov. 2009.
  - [4] E. S. Raymond, "The cathedral and the bazaar," in *The Cathedral and the Bazaar*. O'Reilly and Associates, Oct. 1999.
  - [5] J. Feller and B. Fitzgerald, "A framework analysis of the open source software development paradigm," in *Proceedings of the International Conference on Information Systems*. Association for Information Systems, 2000, pp. 58–69.
  - [6] A. Mockus, R. T. Fielding, and J. Herbsleb, "A case study of open source software development: The Apache server," in *Proceedings of the 2000 International Conference on Software Engineering*, Limerick, Ireland, Jun. 2000, pp. 263–272.
  - [7] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 3, pp. 309–346, Jul. 2002.
  - [8] C. R. Reis and R. P. de Mattos Fortes, "An overview of the software engineering process in the Mozilla project," in *Proceedings of the Open Source Software Development Workshop*, Newcastle upon Tyne, UK, Feb. 2002.
  - [9] S. R. Schach, B. Jin, D. R. Wright, G. Z. Heller, and A. J. Offutt, "Maintainability of the Linux kernel," *IEEE Proceedings – Software*, vol. 149, no. 1, Feb. 2002.
  - [10] L. Yu, S. R. Schach, K. Chen, and J. Offutt, "Categorization of common coupling and its application to the maintainability of the Linux kernel," *IEEE Transactions on Software Engineering*, vol. 30, no. 10, pp. 694–706, 2004.
  - [11] The Apache Software Foundation, "About the Apache HTTP server project," Web page, cited January 16, 2007. [Online]. Available: [http://httpd.apache.org/ABOUT\\_APACHE.html](http://httpd.apache.org/ABOUT_APACHE.html)
  - [12] 104th Congress of the United States, "Health insurance portability and accountability act of 1996," U.S. House of Representatives Report 104-736, 1996. [Online]. Available: <http://www.gpo.gov/fdsys/search/pagedetails.action?granuleId=CRPT-104hrpt736&packageId=CRPT-104hrpt736>
  - [13] B. Kitchenham, "Procedures for performing systematic reviews," Software Engineering Group, Department of Computer Science, Keele University, and Empirical Software Engineering, National ICT Australia Ltd., Tech. Rep. Keele University TR/SE-0401 and NICTA 0400011T.1, Jul. 2004.
  - [14] K. R. Popper, *The Logic of Scientific Discovery*, Routledge Classics ed. London; New York: Routledge, 2002.
  - [15] K. Krippendorff, *Content Analysis: An Introduction to Its Methodology*, 2nd ed. Sage Publications, 2004.
  - [16] F. Trotter, M. Roller, and I. Valdes, "OpenEMR review," Entry in GPL Medicine Wiki, accessed 4 September, 2009, 14 Apr. 2008. [Online]. Available: [http://ehr.gplmedicine.org/index.php/OpenEMR\\_Review](http://ehr.gplmedicine.org/index.php/OpenEMR_Review)
  - [17] Sunset Systems, "About OpenEMR," Web page containing general overview of OpenEMR, from a consultant's web site; accessed 4 September, 2009. [Online]. Available: <http://www.sunsetsystems.com/node/3>
  - [18] P. Burnard, "A method of analysing interview transcripts in qualitative research," *Nurse Education Today*, vol. 11, pp. 461–466, 1991.
  - [19] M. E. Dewey, "Coefficients of agreement," *British Journal of Psychiatry*, vol. 143, pp. 487–489, 1983.
  - [20] R. Bakeman, "Behavioral observation and coding," in *Handbook of research methods in social and personality psychology*, H. T. Reis and C. M. Judge, Eds. New York: Cambridge University Press, 2000, pp. 138–159.
  - [21] J. R. Landis and G. G. Koch, "An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers," *Biometrics*, vol. 33, no. 2, pp. 363–374, Jun. 1977.
  - [22] K. Crowston, H. Annabi, and J. Howison, "Defining open source project success," in *Proceedings of the International Conference on Information Systems*, Dec. 2003, pp. 325–340.
  - [23] M. Donahue, "OpenEMR and its community," Entry in GPL Medicine Wiki, accessed 4 September, 2009, 9 May 2006. [Online]. Available: [http://ehr.gplmedicine.org/index.php/OpenEMR\\_and\\_its\\_community\\_-\\_Margaret\\_Donahue](http://ehr.gplmedicine.org/index.php/OpenEMR_and_its_community_-_Margaret_Donahue)
  - [24] K. Henningson and C. Wohlin, "Assuring fault classification agreement - an empirical evaluation," in *International Symposium on Empirical Software Engineering (ISESE '04)*, 19–20 Aug. 2004, pp. 95–104.
  - [25] K. El Emam and I. Wiczorek, "The repeatability of code defect classifications," in *Proceedings, Ninth International Symposium on Software Reliability Engineering*, Nov. 1998, pp. 322–333.
  - [26] T. Hall, D. Bowes, G. Leibchen, and P. Wernick, "Evaluating three approaches to extracting fault data from software change repositories," in *Product-Focused Software Process Improvement*, ser. Lecture Notes in Computer Science, M. A. Babar, M. Vierimaa, and M. Oivo, Eds. Springer Berlin/Heidelberg, 2010, vol. 6156, pp. 107–115.
  - [27] P. Fusaro, K. El Emam, and B. Smith, "Evaluating the interrater agreement of process capability ratings," in *Proceedings, Fourth International Software Metrics Symposium*, Nov. 1997, pp. 2–11.
  - [28] K. El Emam, D. Goldenson, L. Briand, and P. Marshall, "Interrater agreement in spice-based assessments: some preliminary results," in *Proceedings, Fourth International Conference on the Software Process*, Dec. 1996, pp. 149–156.
  - [29] K. El Emam, J. M. Simon, S. Rousseau, and E. Jacquet, "Cost implications of interrater agreement for software process assessments," in *Proceedings, Fifth International Software Metrics Symposium*, Nov. 1998, pp. 38–51.
  - [30] H.-Y. Lee, H.-W. Jung, C.-S. Chung, J. M. Lee, K. W. Lee, and H. J. Jeong, "Analysis of interrater agreement in iso/iec 15504-based software process assessment," in *Proceedings Second Asia-Pacific Conference on Quality Software, 2001.*, 2001, pp. 341–348.
  - [31] H.-M. Park and H.-W. Jung, "Evaluating interrater agreement with intraclass correlation coefficient in spice-based software process assessment," in *Proceedings, Third International Conference on Quality Software*, Nov. 2003, pp. 308–314.
  - [32] S. Vieira, U. Kaymak, and J. Sousa, "Cohen's kappa coefficient as a performance measure for feature selection," in *2010 IEEE International Conference on Fuzzy Systems (FUZZ)*, 2010, pp. 1–8.
  - [33] M. Mantyla, "An experiment on subjective evolvability evaluation of object-oriented software: explaining factors and interrater agreement," in *International Symposium on Empirical Software Engineering*, Nov. 2005.
  - [34] S. G. Vilbergsdttir, E. T. Hvannberg, and L.-C. Law, "Classification of usability problems (CUP) scheme," in *Proceedings of the 4th Nordic conference on Human-computer interaction (NordicCHI '06)*, 2003, pp. 655–662.