

On the perception of software quality requirements during the project lifecycle

Neil A. Ernst and John Mylopoulos

Department of Computer Science
University of Toronto
Toronto, ON, Canada
{nernerst,jm}@cs.toronto.edu

Abstract. [Context and motivation] A key requirements consideration in software development is the system’s quality requirements. Quality is usually defined in terms of global properties for a software system, such as “reliability”, “usability” and “maintainability”. In the context of software maintenance they are particularly relevant: maintenance activities are performed to ensure software quality. [Question/problem] Recently an expanded view of RE has been emerging, wherein requirements artifacts play a role throughout a system’s lifecycle. How important are quality requirements as the lifecycle progresses? We examine two questions: whether requirements are discussed more as the software matures; secondly, whether different software projects have similar levels of interest about quality requirements. [Principal ideas/results] We use a software repository mining technique we call signifier extraction, and empirically investigate the treatment of software quality in software projects. Signifiers are keywords about quality requirements that we generate using a controlled taxonomy based on ISO9126. Using source data extracted from eight open-source software projects we extract the signifier frequencies over weekly intervals. We analyze the signifier occurrence patterns statistically and historically. [Contribution] Our results show that quality requirements are discussed differently in different projects. Furthermore, there is no correlation between project age and the importance of software quality requirements. Finally, we show that these occurrences provide a roadmap to reconstruct the historical changes of qualities as responses to external forces, such as release cycles and usability audits.

Keywords: Evolution, software quality requirements, repository mining

1 Introduction

Software quality requirements are a key concern throughout the software lifecycle. Requirements research is increasingly focused on supporting systems beyond the initial design phase, captured by Finkelstein’s term ‘reflective requirements’ [6]. Quality requirements are usually defined in terms of global properties

for a software system, such as “reliability”, “usability” and “maintainability”; we think of them as describing the ‘how’, rather than the ‘what’. In this sense “functionality” can also be considered a quality, insofar as it describes how well a given artifact implements a particular function (such as security). The importance of quality requirements lies in their inter-system comparability. Because of their global nature, quality requirements are hard to build into a design and are often treated *post facto* in terms of metrics that are applied to the final product.

If requirements are important throughout the life-cycle (and we believe strongly that they are), a better understanding of requirements after the initial release is important. Are requirements discussed post-release? One way of answering this question is to examine current practices using a standardized requirements taxonomy. In particular, we are interested in finding out whether there is any noticeable pattern in how software project participants conceive of quality requirements. Our study is conducted from the perspective of project participants (e.g., developers, bug reporters, users). We use a set of eight open-source software (OSS) products to test two specific questions about software quality requirements. The first is whether software quality requirements are of more interest as a project ages, as predicted in Lehman’s ‘Seventh Law’ that “the quality of systems will appear to be declining unless they are rigorously maintained and adapted to environmental changes [14, p. 21].” Our second question is whether quality is of similar concern among different projects. That is, is a quality such as *Usability* as important to one project’s participants as it is to another?

To assess these questions, we need to define what we mean by software quality requirements. Our position is that requirements for software quality can be conceived as a set of labels assigned to the conversations of project participants. These conversations take the form of mailing list discussions, bug reports, and commit logs. Consider two developers in an OSS project who are concerned about the software’s performance. To capture this quality requirement, we look for indicators, which we call signifiers, which manifest the concern. We then label the conversations with the appropriate software quality, using text analysis. Our qualities are derived from a standard taxonomy—the ISO 9126-1 software quality model [12]. The signifiers are keywords that are associated with a particular quality. For example, we label a bug report mentioning the **slow** response time of a media player with the *Efficiency* quality.

We discuss related approaches in Section 2. Section 3 describes how we derive these signifiers and how we built our corpora and toolset for extracting the signifiers. We then present our observations and a discussion about significance in Section 4. Finally, we examine some threats to our approach and discuss future work.

2 Related work

Part of our effort with this project is to understand the qualitative and intentional aspects of requirements in software evolution, a notion we first discussed in [9]. That idea is derived from work on narratives of software systems shown in academic work like [1].

Cleland-Huang and her colleagues published work on mining requirements documents for non-functional requirements (quality requirements) in [8]. One approach they tried was similar to this one, with keywords mined from NFR catalogues found in [7]. They managed recall of 80% with precision of 57% for the Security NFR, but could not find a reliable source of keywords for other NFRs. Instead, they developed a supervised classifier by using human experts to identify an NFR training set. There are several reasons we did not follow this route. One, we believe we have a more comprehensive set of terms due to the taxonomy we chose. Secondly, we wanted to compare across projects. Their technique was not compared across different projects and the applicability of the training set to different corpora is unclear. A common taxonomy allows us to make inter-project comparison (subject to the assumption that all projects conceive of these terms in the same way). Finally, the source text we use is less structured than their requirements documents.

Massey [15] and Scacchi ([18, 19]) looked at the topic of requirements in open-source software. Their work discusses the source of the requirements and how they are used in the development process. German [11] looked at GNOME specifically, and listed several sources for requirements: leader interest, mimicry, brainstorming, and prototypes. None of this work addressed quality requirements in OSS, nor did it examine requirements trends.

The difference between projects in level of interest in particular quality requirements was detailed in several case studies described in [?], which describes a methodology which starts with ISO9126 and ‘tailors’ the requirements analysis to specific NFRs.

3 Methodology

Overview: We first construct a set of signifiers, which produces a word list to extensionally define the software quality of interest, e.g., *Efficiency*. We then query corpora from each project with these lists to identify events. Events are timestamped occurrences of our signifiers in the corpora.

3.1 Step I Establishing the corpora

Our corpora are from a selection of eight Gnome projects, listed in Table 1. Gnome is an OSS project that provides a unified desktop environment for Linux and its cousins. Gnome is both a project and an ecosystem: while there are co-ordinated releases, each project operates somewhat independently. In 2002, Koch and Schneider [13] listed 52 developers as being responsible for 80% of the Gnome source code. In our study, the number of contributors is likely higher, since it is easier to participate via email (e.g., feature requests) or bug reports. For example, in Nautilus, there were approximately 2,000 people active on the mailing list, whereas there were 312 committers to the source repository.¹

The projects used in this paper were selected to represent a variety of lifespans and codebase sizes (generated with [20]). All projects were written in C/C++,

¹ Generated using Libresoft, tools.libresoft.es

Table 1. Selected Gnome ecosystem products (*ksloc* = thousand source lines of code)

Product	Language	Size (<i>ksloc</i>)	Age (<i>years</i>)	Type
Evolution	C	313	10.75	Mail
Nautilus	C	108	10.75	File mgr
Metacity	C	66	7.5	Window mgr
Ekiga	C++	54	7	VOIP
Totem	C	49	6.33	Media
Deskbar	Python	21	3.2	Widgets/UI
Evince	C	66	9.75	Doc viewer
Empathy	C	55	1.5	IM

save for one in Python (Deskbar). For each project we created a corpus from that project’s mailing list, subversion logs and the bug comments, as of November 2008. From the corpus, we extracted ‘messages’, that is, the origin, date, and text (e.g, the content of the bug comment), and placed this information into a MySQL database. A message consists of a single bug report, a single email message, or a single commit. If a discussion takes place via email, each individual message about that subject is treated separately. Our dataset consists of over nine hundred thousand such messages, across all eight projects. We do not extract information on the mood of a message: i.e., we cannot tell whether this message expressed a positive attitude towards the requirement in question (e.g., “This menu is unusable”). Furthermore, we are not linking these messages to the implementation in code; we have no way of telling to what extent the code met the requirement beyond participant comments.

Table 2. Qualities and quality signifiers – Wordnet version (WN). Bold text indicates the word appears in ISO/IEC 9126.

Quality	Signifiers
<i>Maintainability</i>	testability changeability analyzability stability maintainability maintain maintainable modularity modifiability understandability
<i>Functionality</i>	security compliance accuracy interoperability suitability functional practicality functionality
<i>Portability</i>	conformance adaptability replaceability installability portable movableness movability portability
<i>Efficiency</i>	resource behaviour time behaviour efficient efficiency
<i>Usability</i>	operability understandability learnability useable usable serviceable usefulness utility useableness usableness serviceableness serviceability usability
<i>Reliability</i>	fault tolerance recoverability maturity reliable dependable responsibility responsibility reliableness reliability dependableness dependability

3.2 Step II Defining qualities with signifiers

In semiotics, Peirce drew a distinction between signifier, signified, and sign [2]. In this work, we make use of signifiers—words like ‘usability’ and ‘usable’—to capture the occurrence in our corpora of the signified—in this example, the concept *Usability*. We extract our signified, concept words from the ISO 9126 quality model [12], which describes six high-level quality requirements (listed in Table 2). There is some debate about the significance and importance of the terms in this model. However, it is “an international standard and thus provides an internationally accepted terminology for software quality [3, p. 58],” which is sufficient for the purposes of this research.

We want to preserve domain-independence, such that we can use the same set of signifiers on different projects. This is why we eschew more conventional text-mining techniques that generate keyword vectors from a training set.

We generate the initial signifiers from Wordnet [10], an English-language ‘lexical database’ that contains semantic relations between words, including meronymy and synonymy. We extract signifiers using Wordnet’s synsets, hypernyms, and related forms (stems), and related components using the two-level hierarchy in ISO9126. When we account for spelling variations, we associate this wordlist with a top-level quality, and use that to find unique events. This gives us a repeatable procedure for each signified quality. We call this initial set of signifiers WN.

Expanding the signifiers – The members of the set of signifiers will have a big effect on the number of events returned. For example, the term ‘user friendly’ is one most would agree is relevant to discussion of usability. However, this term does not appear in Wordnet. To see what effect an expanded list of signifiers would have, we generated a second set (henceforth *ext*), by expanding WN with more software-specific signifiers. The *ext* signifier sets are shown in Table 3.

To construct our expanded sets, we first used Boehm’s 1976 software quality model [4], and classified his eleven ‘ilities’ into their respective ISO9126 qualities. We did the same for the quality model produced by McCall et al. [16]. Finally, we analyzed two mailing lists from the KDE project to enhance the specificity of the sets. Like Gnome, KDE is an open-source desktop suite for Linux, and likely uses comparable terminology. We selected KDE-Usability, which focuses on usability discussions for KDE as a whole; and KDE-Konqueror, a list about a long-lived web browser project. For each high-level quality in ISO9126, we first searched for our existing (WN) signifiers; we then randomly sampled twenty-five mail messages that were relevant to that quality, and selected co-occurring terms relevant to that quality. For example, we add the term ‘performance’ to the synonyms for *Efficiency*, since this term occurs in most mail messages that discuss efficiency.

There are many other possible sources for quality signifiers, but for comparative purposes with the Wordnet lists, we felt these sources were sufficient.

We discuss the differences the two sets create in Section 4.

Table 3. Qualities and quality signifiers – extended version (*ext*). Each quality consists of terms in (a) in addition to the ones listed.

Quality	Signifiers
<i>Maintainability</i>	WN + interdependent dependency encapsulation decentralized modular
<i>Functionality</i>	WN + compliant exploit certificate secured buffer overflow policy malicious trustworthy vulnerable vulnerability accurate secure vulnerability correctness accuracy
<i>Portability</i>	WN + specification migration standardized l10n localization i18n internationalization documentation interoperability transferability
<i>Efficiency</i>	WN + performance profile optimize sluggish factor penalty slower faster slow fast optimization
<i>Usability</i>	WN + gui accessibility menu configure convention standard feature focus ui mouse icons ugly dialog guidelines click default human convention friendly user screen interface flexibility
<i>Reliability</i>	WN + resilience integrity stability stable crash bug fails redundancy error failure

3.3 Step III Querying the corpora

Once we constructed our sets of signifiers, we applied them to the message corpora (the mailing lists, bug trackers, and repositories) to create a table of events. An event is any message (row) in the corpus table which contains at least one term in the signifier set. A message can contain signifiers for different qualities, and can thus generate as many as six events (e.g., a message about maintainability and reliability). However, multiple signifiers for the *same* quality only generate a single event for that quality. We produced a set of events (e.g., a subversion commit message), along with the associated time and project. We group events by week for scalability reasons. Note that each email message in a thread constitutes a single event. This means that it is possible that a single mention of a signifier in the original message might be replied to multiple times. We assume these replies are ‘on-topic’ and related to the original concern.

We normalize the extracted event counts to remove the effect of changes in mailing list volume or commit log activity (some projects are much more active). The calculation takes each signifier’s event count for that period, and divides by the overall number of messages in the same period. We also remove low-volume periods from consideration. This is because a week in which only one message appeared, that contained a signifier, will present as a 100% match. From this dataset we conducted our observations and statistical tests. Table 4 illustrates some of the sample events we dealt with, and our subsequent mapping to software quality requirements.

3.4 Step IV Precision and recall

We verified the percentage of terms retrieved that were unrelated to a signified software quality to understand the precision of our method. For example, we

Table 4. Classification examples. Signifiers causing a match are highlighted.

Event	Quality
...By upgrading to a newer version of GNOME you could receive bug fixes and new functionality.	<i>None</i>
There should be a feature added that allows you to keep the current functionality for those on workstations (automatic hot-sync) and then another option that allows you to manually initiate .	<i>Functionality</i>
Steps to reproduce the crash : 1. Can't reproduce with accuracy . Seemingly random.	<i>Reliability, Functionality</i>
How do we go disabling ekiga's dependency on these functions, so that people who arn't using linux can build the program without having to resort to open heart surgery on the code?	<i>Maintainability</i>
U_() is equivalent of _() but returns Unicode (UTF-8) string. Update your xml- i18n -tools from CVS (recent version understands U_), update Swedish translation and close the bug back.	<i>Portability</i>
On some thought, centering dialogs on the panel seems like it's probably right, assuming we keep the dialog on the screen, which should happen with latest metacity.	<i>Usability</i>
These calls are just a waste of time for client and server, and the Nautilus online storage view is slowed down by this wastefulness.	<i>Efficiency</i>

encountered some mail messages from individuals whose email signature included the words "Usability Engineer". If the body of the message wasn't obviously about usability, we coded this as a false-positive. Our error test was to randomly select messages from the corpora and code them as relevant or irrelevant. We assessed 100 events per quality, for each set of signifiers (*ext* and *WN*). Table 5 presents the results of this test. False-positives averaged 21% and 20% of events, for *ext* and *WN* respectively (i.e., precision was 79% and 80%).

Recall, or completeness, is defined as the number of relevant events retrieved divided by the total number of relevant events. Superficially we could describe our recall as 100%, since the query engine returns all matches we asked for, but true recall should be calculated using all events that had that quality as a topic. To assess this, we randomly sampled our corpora and classified each event into either a signifier (*Usability*, *Reliability*, etc.) or *None*. For extended signifier lists, we had an overall recall of 51%, and a poor 6% recall for the Wordnet signifiers. We therefore dispensed with the Wordnet signifiers. This is a very subjective process. For example, we classified a third of the events as *None*; however, arguably any discussion of software could be related, albeit tangentially, to an ISO9126 quality. We think a better understanding of this issue is more properly suited to a qualitative study, in which project-specific quality models can be best established.

4 Observations and discussion

This section first explains the frequency distributions of the data we collected. We then use that data to answer the two questions raised in the introduction:

Table 5. False positive rates for the two signifier sets

Signified quality	F.P. Rate	ext F.P. Rate WN
Usability	0.47	0.22
Portability	0.11	0.20
Maintainability	0.22	0.31
Reliability	0.15	0.19
Functionality	0.14	0.18
Efficiency	0.16	0.07
Mean	0.21	0.20

- 1) Is there a correlation between discussion of quality requirements and project age?
- 2) Are quality requirements of similar importance relative to each project?

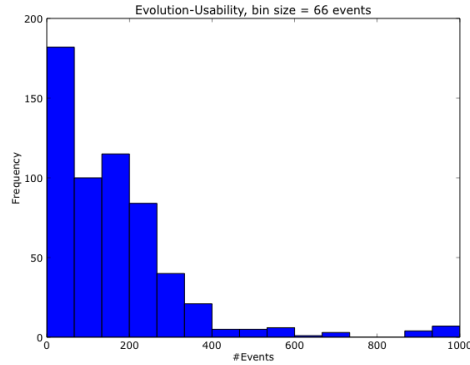


Fig. 1. Frequency distribution for Evolution-Usability. *x-axis* represents number of events (66 events wide), *y-axis* the number of weeks in that bin.

4.1 Data distribution

Fig. 1 shows an example frequency distribution for the quality requirement *Usability*, product *Evolution*, with non-normalized data. The distributions seem to follow a power-law distribution, that is, a majority of weeks had few events, with the ‘long tail’ consisting of those weeks with many events. We verified that this pattern also existed for the remaining qualities and project combinations.

4.2 Examining quality discussions over time

Our first question was whether, as predicted in the literature, there was a correlation between the importance of software quality requirement and the age of a project. We examined this in three ways. First, we looked at the overall trends for a project. Secondly, we used release windows, the time between the release

Table 6. Selected summary statistics, normalized. Examples from Nautilus and Evolution for all qualities using extended signifiers.

Project	Quality	r^2	slope	N (weeks)
Evolution	Efficiency	0.06	-0.02	439
	Portability	0.08	-0.05	448
	Maintainability	0.04	-0.02	320
	Reliability	0.20	0.25	492
	Functionality	0.03	-0.02	439
	Usability	0.14	0.27	515
Nautilus	Efficiency	0.16	-0.10	420
	Portability	0.16	-0.07	331
	Maintainability	0.27	-0.09	216
	Reliability	0.19	0.26	454
	Functionality	0.12	-0.05	390
	Usability	0.08	0.29	459

of one version, and the release of the next (major) version. Finally, we explored qualitative explanations for patterns in the data.

Using project lifespan – We examined whether, over a project’s complete lifespan, there was a correlation with quality event occurrences. Recall that we define quality events as occurrences of a quality signifier in a message in the corpora. We performed a linear regression analysis and generated correlation coefficients for all eight projects and six qualities. Figure 2 is an example of our analysis. It is a scatterplot of quality events vs. time for the *Usability* quality in Evolution. For example, in 2000/2001, there is a cluster around the 300 mark, using the extended (ext) set of signifiers. Note that the y-axis is in units of (events/volume * 1000) for readability reasons.

The straight line is a linear regression. The dashed vertical lines represent Gnome project milestones, with which the release dates of the projects we study are synchronized. Release numbers are listed next to the dashed lines. Due to

Table 7. Selected summary statistics, normalized. Examples from *Usability* and *Efficiency* (performance) for selected products using extended signifiers.

Quality	Project	r^2	slope	N (weeks)
Usability	Deskbar	0.08	-0.97	126
	Evolution	0.14	0.27	515
	Nautilus	0.08	0.29	459
	Totem	0.20	0.63	314
Efficiency	Deskbar	0.00	-0.11	34
	Evolution	0.06	-0.02	439
	Nautilus	0.16	-0.10	420
	Totem	0.10	-0.16	158

space constraints, Table 6 lists only Nautilus and Evolution as products, and r^2 squared correlation value, or coefficient of determination and slope (trend) values for each quality within that project. r^2 varies between 0 and 1, with a value of 1 indicating perfect correlation. The sign of the slope value indicates direction of the trend. A negative slope would imply a decreasing number of occurrences as the project ages. Table 7 does a similar analysis for all products and the *Usability* and *Efficiency* (performance) qualities.

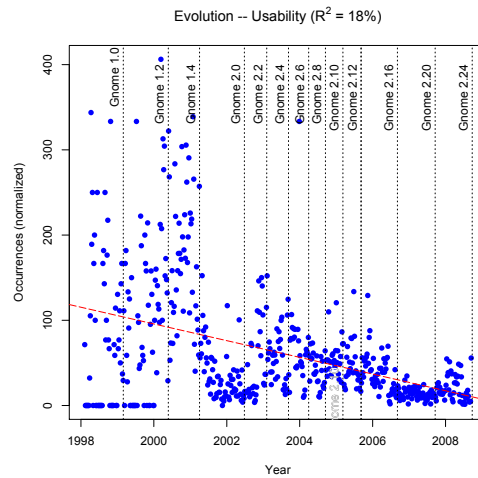


Fig. 2. Signifier occurrences per week, Evolution – *Usability*

The results are inconclusive. In all cases the correlation coefficient indicating the explanatory power of our linear regression model is quite low, well below the 0.9 threshold used in, for example, [17]. There does not seem to be any reason to move to non-linear regression models based on the data analysis we performed. We conclude that our extended list of signifiers does not provide any evidence of a relationship between discussions of software quality requirements and time. In other words, either the occurrences of our signifiers are random, or there is a pattern, and our signifier lists are not adequately capturing it. The former conclusion seems more likely based on our inspection of the data.

Using release windows – It is possible that the event occurrences are more strongly correlated with time periods prior to a major release, that is, that there is some cyclical or autocorrelated pattern in the data. We defined a release window as the period from immediately after a release to just before the next release. We investigated whether there was a higher degree of correlation between the number of quality events and release age, for selected projects and keywords. Was this release window correlation better than the one we found for project lifespan as a whole? For space reasons we do not include these results, but there was

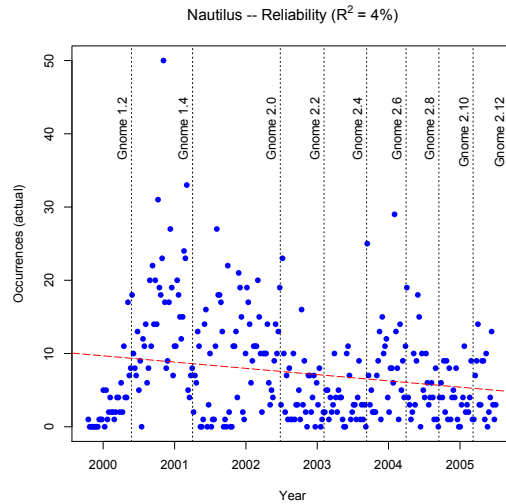


Fig. 3. Signifier occurrences per week, Nautilus – *Reliability*

no improvement in correlation. There is no relationship between an approaching release date and an increasing interest in software quality requirements.

Analysis of key peaks in selected graphs – The final explanation we explore is that the data are unrelated to software age or release cycle, and are instead responding to external events, such as a usability audit. We chose to look at Evolution, a mail and calendar client, and Nautilus, a web browser and file manager, for more detailed ‘historical’ analysis. We tried two approaches: one used the normalized data, and identified periods where our signifier occurred more frequently with respect to everyday volume. The second approach used the actual signifier counts to see why that signifier occurred more frequently than other periods.

We looked at the normalized *Usability* events in Evolution, shown in Fig. 2. To eliminate bug reports and triaging events, we excluded these types of data from our query. Many bug reports are auto-generated, and contribute more noise than signal. For instance, one initial peak we examined was related to the “Mass close of stale bugs $i = 4$ months old.” This generated a lot of noise as the signifiers in these reports are considered once more by our algorithm (since we treat any discussion on a bug similarly to mail threads).

With these events removed as noise, Fig. 2 shows a cluster of points in early 2000. Mailing list discussions at that time turned to a question about the default option for forwarding mail messages, e.g., “... I know this was discussed a few weeks ago ... could it be implemented as an advanced option that has to be turned on and is off by default?” Later that year, in October, another spike in our graph can be attributed to a feature-freeze on Evolution and associated UI cleanups. As Evolution 1.4 is released in mid-2003, there is a small upward trend. Events at that time reflect problems with the new release, reflecting some

UI changes. We still see some effects due to volume, such as the outlier near the end of 2003, where nearly one third of mailing messages were usability related. The issue here is one of overall volume over the winter holidays. In this case a single mail thread about keyboard shortcuts consumed the discussions.

For our second approach, we used the actual signifier event counts, and targeted *Reliability* events for Nautilus. In November, 2000, 50 events occur. Inspecting the events, one can see that a number have to do with bug testing the second preview release that was released a few days prior. For example, one event mentions ways to verify reliability requirements using hourly builds: “As a result, you may encounter a number of **bugs** that have already been fixed. So, if you plan to submit **bug** reports, it’s especially important to have a correct installation!”. Secondly, in early 2004 there is a point with 29 events just prior to the release of Gnome 2.6. Discussion centers around the proper treatment of file types that respects reliability requirements. It is not clear whether these discussions are in response to the external pressure of the deadline or are just part of a general, if heated, discussion.

These investigations show that there is value to examining the historical record of a project in detail, beyond quantitative analysis. While some events are clearly responding to external pressures such as release deadlines, other events are often prompted by something as simple as participant interest, which seems to be central to the OSS development model.

4.3 Quality importance and project

Table 8. Quality per project. Numbers indicate normalized occurrences per week.

Quality	Project	Occurrences
Efficiency	Evolution	0.012
	Nautilus	0.026
Usability	Evolution	0.192
	Nautilus	0.285
Portability	Evolution	0.010
	Nautilus	0.011

Recall that in our second question, we wanted to examine whether certain projects would be more concerned with software quality requirements than others. We characterized the importance of a requirement to a project by calculating the mean normalized occurrences of the signifier (such as *Usability*) over time. This controls for both project longevity and project size.

Table 8 lists our results; for space considerations, only three (representative) qualities are listed. We show the mean number of occurrences per week, normalized by dividing by the overall number of ‘events’ in that period, to eliminate the effect of volume. We would like to know, in other words, what proportion of all messages in that week were talking about the requirement of interest.

We used the extended signifier set (ext). We cannot compare between qualities, because the signifier sets are not the same size. However, there is a difference among projects. We chose to focus on Nautilus and Evolution (both projects of similar longevity, focused on file management and mail respectively). The *Efficiency* quality occurs in Nautilus discussions at a rate of 0.026 occurrences per week, and in Evolution at 0.012 occurrences per week – less than half as often. *Usability* is discussed 1.5 times as often in Nautilus, while other requirements, including *Portability*, show no difference. One possible explanation is that Evolution participants have a conceptual model of Efficiency that is a poorer fit to our signifier lists than the model Nautilus participants use. However, it does seem fair to conclude that projects have different interests with respect to software quality. We intend to do further testing to explore how communities conceptualize these fairly abstract ‘-ilities’.

4.4 Threats to validity

Construct validity The main threat to construct validity is that our signifiers may omit relevant terms or phrases., e.g., “can’t find the submit button” vs. “usability”. Our qualities are not directly comparable, since their respective signifier set sizes differ. *Usability*, for example, has 24 terms in its bubble, versus *Functionality* with 10. We conducted the error analysis to determine how accurate our bubbles are. Our error validation should be conducted by more people to ensure inter-rater reliability. Many events are tricky to classify. Furthermore, we are assuming that projects share the ontology of software quality expressed in the quality model (ISO9126). A more domain-specific taxonomy would be useful.

Internal validity When we perform our regression analysis, assuming a linear relationship may not be a good model of the actual pattern these discussions follow. We focused on the linear model as it is the simplest explanation of the pattern we would expect to see if quality discussions were increasing with time. Our source data may not capture all discussions regarding quality requirements – we omitted IRC chats, for instance. However, these data sources are most amenable to large-scale analysis. Follow-up with qualitative studies would be useful.

External validity Our data originated from open-source projects, less than ten years old, from the Gnome ecosystem. Of these, the open-source nature of the project seems most problematic for external validity. Capra et al. [5], for example, show a higher software quality in OSS projects than commercial projects. It would be interesting to determine whether a top-down directive to focus on software quality, or some other methodological change, would present as a noticeable spike on the event occurrence graph.

4.5 Models of quality requirements

There is a rich history of discussion regarding software quality requirements, and quality models in particular. The main problem that arose in our study was that the quality models are (by design) very high-level. They provide a useful baseline from which to derive more specific models. However, it is useful to have a

cross-product quality requirements model which can be used to compare software systems. Many questions are left unanswered when confronted by actual data: for instance, what is the relationship between product reliability and product functionality?

The challenge for researchers is to align software quality models, at the high level, with the product-specific requirements models developers and community participants work with, even if these models are implicit. One reason discussions of quality requirements were difficult to identify is that, without explicit models, these requirements are not properly considered or are applied haphazardly. We need to establish a mapping between the platonic ideal and the reality on the ground. This will allow us to compare maintenance strategies for product quality requirements across domains, to see whether strategies in, for example, Gnome, can be translated to KDE, Apple, or Windows software.

5 Conclusions and future work

This paper presents a novel analysis technique for conducting empirical research in Requirements Engineering. The technique has been applied to study two specific questions concerning quality requirements. In accordance with Lehman's laws of software evolution, we hypothesized that there is growing interest in quality requirements within a developer community as a project matures. However, our analysis provides no evidence for this hypothesis. However, it is sometimes possible to use external events to explain patterns in the data. We then showed that there is a difference in how different projects treat software qualities with some projects discussing certain quality requirements more than others. We have not presented a formal hypothesis about what this might suggest.

Our ultimate goal is to be able to extract, from available sources, a list of requirements for a project, so that we can trace not just the 'physical' changes in the codebase, but also the evolving features and goals inherent in a project. We plan to continue our experiments with repository mining with this in mind. We have begun work using multi-label classifiers on more domain-specific taxonomies (e.g., database systems). We think 'ground-truthing' our results with qualitative studies would be useful to make our results inform a theory about quality in software, such that techniques could be predictive as well as descriptive.

6 Appendix and acknowledgements

We appreciate the comments of the software engineering group at the University of Toronto and Abram Hindle, and the comments of anonymous reviewers. Source code, processed data, and related discussions are available at <http://neilernst.net/tag/msr/>.

References

1. Antón, A.I., Potts, C.: Functional paleontology: system evolution as the user sees it. In: International Conference Software Engineering. pp. 421–430. Toronto, Canada (2001)

2. Atkin, A.: Peirce's Theory of Signs (October 2006), <http://plato.stanford.edu/entries/peirce-semiotics/>
3. Bøegh, J.: A New Standard for Quality Requirements. *IEEE Soft.* 25(2), 57–63 (2008)
4. Boehm, B., Brown, J.R., Lipow, M.: Quantitative Evaluation of Software Quality. In: *International Conference Software Engineering*. pp. 592–605 (1976)
5. Capra, E., Francalanci, C., Merlo, F.: An Empirical Study on the Relationship Between Software Design Quality, Development Effort and Governance in Open Source Projects. *Trans. Soft. Eng.* (2008)
6. Cheng, B.H., de Lemos, R., Giese, H., Inverardi, P., Magee, J.: *Software Engineering for Self-Adaptive Systems: A Research Roadmap*, pp. 1–26. Springer-Verlag, LNCS 5525 (2009)
7. Chung, L., Nixon, B.A., Yu, E.S., Mylopoulos, J.: *Non-Functional Requirements in Software Engineering*, *International Series in Software Engineering*, vol. 5. Kluwer Academic Publishers, Boston (October 1999)
8. Cleland-Huang, J., Settini, R., Zou, X., Solc, P.: The Detection and Classification of Non-Functional Requirements with Application to Early Aspects. In: *International Conference Requirements Engineering*. pp. 39–48. IEEE, Minneapolis, Minnesota (2006)
9. Ernst, N.A., Mylopoulos, J.: Tracing software evolution history with design goals. In: *International Workshop on Software Evolvability at ICSM*. Paris, France (October 2007)
10. Fellbaum, C. (ed.): *WordNet: An Electronic Lexical Database*. MIT Press (1998)
11. German, D.M.: The GNOME project: a case study of open source, global software development. *Soft. Process: Improvement and Practice* 8(4), 201–215 (2003)
12. Coallier, François (ed.). *Software engineering – Product quality – Part 1: Quality model*. Standard ISO9126, International Standards Organization - JTC 1/SC 7 (2001)
13. Koch, S., Schneider, G.: Effort, co-operation and co-ordination in an open source software project: GNOME. *Inf. Sys. J.* 12, 27–42 (2002)
14. Lehman, M.M., Ramil, J.F., Wernick, P.D., Perry, D.E., Turski, W.M.: Metrics and laws of software evolution-the nineties view. In: *International Software Metrics Symposium* pp. 20–32. Albuquerque, NM (1997)
15. Massey, B.: Where Do Open Source Requirements Come From (And What Should We Do About It)? In: *Workshop on Open Source Software Engineering at ICSE*. Orlando, FL, USA (2002)
16. McCall, J.: *Factors in Software Quality: Preliminary Handbook on Software Quality for an Acquisition Manager*, vol. 1-3. General Electric (November 1977)
17. Mens, T., Fernandez-Ramil, J., Degrandart, S.: The evolution of Eclipse. In: *International Conference Software Maintenance*. pp. 386–395. Shanghai, China (October 2008)
18. Scacchi, W.: Understanding the requirements for developing open source softwaresystems. *IET Software* 149(1), 24–39 (2002)
19. Scacchi, W., Jensen, C., Noll, J., Elliott, M.: Multi-Modal Modeling, Analysis and Validation of Open Source Software Requirements Processes. In: *International Conference on Open Source Software*. vol. 1, pp. 1–8. Genoa, Italy (July 2005)
20. Wheeler, D.: SLOCcount (2009), <http://www.dwheeler.com/sloccount/>