

Do Crosscutting Concerns Cause Modularity Problems?

Robert J. Walker, Shreya Rawal, Jonathan Sillito
University of Calgary
Calgary, Canada
{walker,srawal,sillito}@ucalgary.ca

ABSTRACT

It has been claimed that crosscutting concerns are pervasive and problematic, leading to difficulties in program comprehension, evolution, and long-term design degradation. To consider whether this theory bears out, we examine the patch history of the Mozilla project over a period of a decade to consider whether crosscutting concerns exist therein and whether we can see evidence of problems arising from them. Mozilla is an interesting case, due to its longevity; size; polylingual nature; and use of a patch review process, which maintains strong connections between issue reports and the patches that are intended to address each. We perform several statistical analyses of the over 200,000 patches submitted to address over 90,000 issues reported in this time period. We find that 90% of patches show little or no evidence of scattering, that the scattering of a patch tends to decrease slightly upon review on average, and that the system shows at worst a slow increase of average scattering over time.

Categories and Subject Descriptors

D.2.8 [Software Engineering]: Metrics; D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement

General Terms

Experimentation, design, measurement, verification.

Keywords

Scattering, AOSD, retroactive study, Mozilla, power laws.

1. INTRODUCTION

It has been argued for decades that software should be well-modularized to obtain various benefits, such as ease of change, ease of comprehension, and ease of parallel development [32]. In the 1990s, it was understood that there exist software concerns (known as *crosscutting concerns*) that

fail to be encapsulated by the otherwise good modularization present in real systems [20], but are instead *scattered* across and *tangled* with the other, modularized concerns in a system; typical examples include logging, security, distribution, protocol enforcement, and various design patterns. Furthermore, it was argued that crosscutting concerns were inherent, and not an artifact of a poorly-chosen modularization [38]. In response, the solution was promoted to separate and encapsulate crosscutting concerns through novel programmatic mechanisms; hence was born the aspect-oriented software development (AOSD) research community. But in considering how the modularity issues should be addressed, a key issue was often overlooked: do crosscutting concerns cause modularity problems in practice?

To be clear, there has been much work that has attempted to examine whether the novel AOSD mechanisms represent an improvement in modularity over traditional mechanisms, but all such work starts from the assumption that crosscutting concerns are necessarily deleterious. We can see several serious threats to validity in such studies. (1) Tiny examples of problems plus assertions that their properties extrapolate, while reasonable at early research stages to explain issues, cannot serve as demonstration that problems exist in practice. (2) Measures of internal quality attributes (e.g., coupling) can be used descriptively to characterize the properties of software. But they cannot be used to demonstrate improvement of external quality attributes (e.g., ease of change, ease of comprehension, and ease of parallel development) in the absence of their independent validation [22, 9]. (3) Similarly, structural or behavioural models can be used to demonstrate improvement only relative to the assumptions of the model. In the absence of generalizable validation of those assumptions, purely model-based claims of improvement constitute circular reasoning. (4) Comparisons between two versions of a program, one written with traditional programming mechanisms and the other with aspect-oriented (AO) programming mechanisms, are potentially biased when the AO version is written with full knowledge of its evolutionary path or the measurements to be made.

We seek then to examine more closely and with a minimum of assumptions whether modularity problems arise in systems using conventional modularization mechanisms (i.e., not AO ones) as predicted by standard aspect-oriented theory. To this end, we analyze the change history for the Mozilla project over a 10 year period of development. Mozilla is an interesting and pertinent case study for crosscutting concerns: it is large (over 11 MLOC spread amongst nearly 100,000 files in a recent snapshot), so it potentially

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGSOFT'12/FSE-20, November 10–18 2012, Cary, NC, USA
Copyright 2012 ACM 978-1-4503-1614-9/12/11 ...\$15.00.

contains many crosscutting concerns; it is mature, so there has been adequate time for problems to develop; it is written in multiple languages and data formats, allowing a more general analysis; and it uses a patch review process that (a) allows a reliable mapping from issue reports to patches that purport to solve the issues and (b) provides a means of seeing rejected attempts at providing patches, potentially providing insight into issues that were difficult to address.

We expect that a patch is scattered (i.e., it includes changes to multiple parts of the codebase) when the underlying issue being addressed relates to crosscutting concerns in the codebase; analyzing the scattering of patches thus gives us an opportunity to understand the impact that scattered concerns have on development work in practice. We focus on scattering rather than tangling, as simple measures of scattering arise directly from the change history, both simplifying the collection process and minimizing the potential for biases arising from tailored measures; to measure tangling would instead require the reconstruction of large numbers of historical snapshots of the codebase, or the use of questionable heuristics.

We have gathered issue, patch, and review information from Mozilla’s issue tracking system to consider the extent, nature, and changes to the scattering of patches. More specifically, we have analyzed this information to address the following research questions: (1) Does Mozilla show signs of crosscutting concerns? (2) Is the scattering of patches spurious? (3) What is the nature of the highly scattered changes? (4) How scattered are the patches? (5) Do scattered patches require more revision? (6) Does scattering increase after review? (7) Does Mozilla show signs of decay? We find that, although crosscutting concerns are clearly present, they show little evidence overall of causing problems in Mozilla.

The remainder of the paper is structured as follows. In Section 2, we examine previous work. In Section 3, we describe some details of the Mozilla project and the data that we collected from it. Details of the analysis and its results are given in Section 4, in which we take an exploratory approach to deriving each research question. Threats to validity of our study are discussed in Section 5. Conclusions about our study are left until Section 6.

This paper contributes an assessment of evidence about the presence and effects of crosscutting concerns within a large-scale industrial system.

2. PREVIOUS WORK

Much work in AOSD has created or applied measures of internal quality attributes (e.g., coupling, entity counts), particularly with an eye to considering how these measures change as the software system evolves, or for comparison between systems employing aspect-oriented and traditional mechanisms [14, 13, 17, 25, 10]. As argued in the introduction, internal quality attributes do not immediately imply anything about the external quality attributes that we care about for modularity: ease of change, comprehension, and parallel development [32]. Przybyłek [33] has argued that typical aspect-oriented measures fail to consider the full range of dependencies in aspect-oriented systems, leading to these measures being biased in favour of AOSD approaches; his alternatives remain internal quality measures.

Stability has been argued as a means for bridging the gap between internal and external quality attributes [40]. Sev-

eral papers have examined the use of stability to evaluate AOSD (e.g., [17, 10]). Kelly [19] conducted a case study to examine how stability can be used to evaluate a design; she explicitly points out that “since stability is not a sufficient criterion, software engineering knowledge must be used to reason about” candidate design characteristics related to long-term maintainability.

Baldwin and Clark [5] propose a method to compare and evaluate the design of a system based on design structure matrices (DSMs), a modelling approach to represent dependencies in a system. Various studies have used DSMs to investigate the modularity of systems (e.g., [37, 23]). In particular, MacCormack et al. [27] used the DSM approach to compare the first version of Mozilla that was released as open source and a subsequent redesign of the codebase and found that the redesigned code base was more modular than its predecessor. Unfortunately, DSMs necessitate significant assumptions about what constitutes a dependency and the impact of a dependency as a system evolves. In contrast, our work aims to understand how the modularity of the system affects the ongoing development of the system by investigating how scattered the changes are that developers make.

Scattering has been measured in several ways. Absolute approaches (i.e., counts of entities affected) [41, 35] often have the advantage that they can avoid biasing the data. Several works use relative approaches (e.g., concern diffusion) [14, 13, 17, 9, 2]; the advantage of these is that scattering can then be compared between systems or versions. We use absolute measures in our study, and through our analysis (see Section 4.7), derive a relative measure in order to compare scattering between different contexts.

Baldi et al. [4] utilized an entropy-based model to analyze the lexical details of software systems to demonstrate the presence of crosscutting concerns in a purely automated manner, while also measuring the degree of scattering and tangling; we do not attempt to identify crosscutting concerns beyond the issues with which they are associated.

2.1 Results Not Supporting AOSD’s Claims

While the majority of the AOSD literature claims improved modularity, some works argue that at best the reality is more nuanced and at worst those claims are false.

Marin et al. [28, 29] classify crosscutting concerns, based on literature review and their own experience; some of their categories appear not to be harmful to software systems [11]. Kienzle and Gélinau [21] performed a case study in which they attempted to modularize transactional properties while ensuring the classic ACID properties from database research; they found that subtle and complex design issues arise that, if not handled well, could lead to poor modularity. Siadat et al. [36] considered whether a variety of optimizations in a network simulator could be modularized through an AO approach; they found that, while maintaining good performance, comprehensibility was clearly negatively affected by the separation. Figueiredo et al. [10] evolved two software product lines to assess their modularity properties; they found that not all crosscutting concerns cause problems, and forcing some crosscutting concerns into separated aspects can itself cause problems. Bartsch and Harrison [6] conducted an experiment in which 11 developers were asked to make modifications to either a Java- or an AspectJ-based implementation of a particular system; they found no statistical improvement for the AspectJ version. Przybyłek [34]

examined ten applications possessing both Java and AspectJ implementations to arrive at the conclusion, “the claim that ‘the software built in AOP is more modular than the software built in OOP’ is a myth.” In fairness, his measures have also not been validated, but clearly, the question as to the effects on modularity by crosscutting concerns have not been settled at this point.

2.2 “Do Crosscutting Concerns Cause Defects?”

An important study to consider here is that of Eaddy et al. [9]. They also point out that effects on internal quality attributes do not signify effects on external quality attributes. They set out to measure whether scattering and defects are correlated, by examining the histories of three systems. Their methodology was complicated by their need to link particular code, particular bugs, and particular concerns, in addition to the need to design and collect a number of complicated measures. In the end, they claim to demonstrate a statistically significant correlation between the scattering of a concern and its likelihood to have defects, independently of the size of the concern’s implementation. Although their study was carefully conducted, their linking methodology is a serious threat to internal validity, as they point out; independent means of investigating crosscutting concerns are warranted.

Figueiredo et al. [12] conducted five controlled experiments to ascertain how well people can map code to concerns; they found that developers tend to underreport which concerns are present in a given piece of code, that all concern-based measures also lead to such biases, but that coarse-grained scattering measures are less sensitive to poor mappings from code to concerns. We take this as support that utilizing simple measures of scattering arising directly from Mozilla’s repository may help avoid biases.

More recently, Aversano et al. [2] also report a correlation between the number of defects in design-pattern classes and the scattering of the induced crosscutting concerns.

2.3 Evolvability in Open Source

Previous work has studied the evolvability and modularization of various open source systems including the Mozilla code base. For example, German et al. [15, 16] have conducted studies of the evolution of the Mozilla, Apache, PostgreSQL, and GCC code bases. They classified modification records (MRs) based on the type of activity they reflect: functional improvement, defect fixing, architectural evolution and refactoring, documentation, etc. Their analysis of 117,554 MRs from the Mozilla project found that on average 3 files were changed per MR [16], suggesting that scattering was not significant. However, measures of central tendency can be poor summaries of many distributions.

3. THE DATA REPOSITORY

The Mozilla project¹ is an open source project responsible for a range of applications, most notably the Firefox web browser, Thunderbird email client, and Bugzilla issue tracking system. Previous research has described the development process followed by participants in the Mozilla project (e.g., [30, 1, 18, 31]). Central to this process is their issue

tracking system (Bugzilla), which tracks issue reports of any type including bug reports and enhancement requests.

Once an issue is reported, that report is used as a discussion forum on whether to make the requested change and how it should be made [7]. A bug that cannot be duplicated or any other issue that is judged as not worth addressing may simply be marked as *closed* and not pursued further. On the other hand, if participants in the community decide to implement a change to address the issue, the issue tracking system is used to facilitate source code review. This review process begins when a developer uploads a *patch* (a file capturing the *diff*s of proposed changes to the code) to the issue tracking system, which is associated with a specific issue report. Other developers can download that patch, review it, and make comments on it. Module owners can also formally mark a patch as approved or not. Mozilla’s review process is often iterative and as problems are identified, a new patch can be prepared and uploaded with the previous patch being marked as obsolete. Multiple patches can simultaneously be non-obsolete; we consider the union of non-obsolete patches in our analyses where appropriate.

3.1 Data

We examined a total of 93,006 issue reports submitted for the Mozilla project. First we used the search and export feature of Bugzilla to generate a list of issue reports. Then we used custom Python scripts to extract the data regarding each issue report, such as the patch developer’s identity, submission date and time stamp, issue description, and attachments. We analyzed a total of 221,508 patches submitted from 8 April 1999 (patch ID 1) to 17 June 2010 (patch ID 452149). In our data set, we had a total of 91,085 obsolete patches and 130,256 non-obsolete patches (there were also 167 patches for which obsolescence could not be determined, and were excluded wherever the question of obsolescence mattered).

Each patch file contains the path of each file affected. We also collected other parameters that were associated with the patch: lines added, removed, and modified; obsolescence; review status; developer of the patch; and the time stamp when the patch was uploaded.

3.2 Our Scattering Measures

There are many potential ways of measuring the amount of scattering of a patch. As mentioned in Sections 1 and 2.2, we are concerned with biasing the results through complex measures. We selected measures for the sake of simplicity and ease of computation from the patch data.

The number of files affected per patch (FPP) is the number of unique files in the codebase that were modified or added as part of the patch for an issue. When a patch file is submitted to the Mozilla issue tracker, the generated *diff* contains the file name of all the files that were changed.

The code for all the individual subprojects in the Mozilla family (e.g., Firefox, Thunderbird, etc.) is combined into a single source tree, subdivided into directories referred to as *modules*. Each module is specific to some particular feature of the project. For example, the `browser` module contains the front-end code for the Firefox browser. The number of modules affected per patch (MPP) is the number of unique modules modified (including additions and deletions) while addressing a change task. MPP has the potential to evaluate the modularity of the directory structure created by

¹<http://www.mozilla.org/>

the Mozilla developers. MPP is calculated from the path information captured by the individual patches.

As mentioned above, multiple patches can exist for the same issue report; a subset of these will be obsolete, and the remainder will be non-obsolete. For some of our analyses, instead of working with per-patch metrics, we union the files affected by each patch for a given issue report; this yields the number of files affected per issue considering only obsolete patches (FPI_{obs}) and the number of files affected per issue considering only non-obsolete patches (FPI_{nob}).

The unioning procedure is ambiguous for modules, as patches often contain path information local to the contributing developer's filesystem, at variance with other patches; we thus do not report modules per issue measurements, to avoid errors that would bias the results.

4. THE ANALYSIS

Our analysis proceeded in an exploratory fashion, discovering new questions while addressing existing ones. Each subsection below describes the detailed research questions that we examined, why the questions matter, how we addressed the questions, and what we found.

4.1 Are there signs of crosscutting concerns?

This question is a fundamental one, and not to be taken lightly. Although suggestions have been made that every system necessarily possesses crosscutting concerns, there are two points to recognize: (a) we should test hypotheses even if there is widespread agreement on them; and (b) if we were unable to see crosscutting concerns through our measurements, our methodology might possess a serious flaw.

As 221,508 patches are a lot to visualize, we need some means of summarizing them. In terms of standard descriptive statistics, for the FPP measure, the mean is 5.09,² the median is 2, the mode is 1, the standard deviation is 12.99, the minimum is 1, and the maximum is 872. For the MPP measure, the mean is 1.48, the median is 1, the mode is 1, and the standard deviation is 1.81, the minimum is 1, and the maximum is 136.

These statistics suggest that indeed, at least some patches are scattered; scattered patches are definite signs of crosscutting concerns (by definition)—*if* the scattering of the patches are reflective of the scattering of the underlying concern. Patch scattering might occur for other, spurious reasons.

4.2 Is the scattering of patches spurious?

Mozilla's patch review process is supposed to catch situations where developers have lumped together unrelated sets of changes as patches; these should be broken up into individual patches, each addressing an individual issue. Undoubtedly, not all patches get reviewed in practice, but more complex ones (like ones involving scattered changes) are unlikely to be ignored by module owners. We note that Ayari et al. [3] conducted a case study to manually inspect issue reports in Mozilla, finding very few cases where changes were made without an issue report; this is reassuring, but does not guarantee that the related patches do not address additional, unrelated issues.

As a simple test, we can examine the patches for issue reports where review has clearly occurred, as evidenced by the

²We report all decimals to 2 digits for convenience.

fact that some are marked as obsolete. If we see that scattering decreases markedly in these cases, this would indicate that unrelated issues are being pushed out of the patches. Of the 93,006 total issue reports that we collected, 34,726 of them display at least one patch marked as obsolete; each of these issue reports also has at least one patch marked as non-obsolete. The standard descriptive statistics for the FPI_{nob} measurements are as follows: the mean is 6.58, the median is 3, the mode is 1, the standard deviation is 16.14, the minimum is 1, and the maximum is 635. The standard descriptive statistics for the FPI_{obs} measurements are as follows: the mean is 7.09, the median is 2, the mode is 1, the standard deviation is 24.32, the minimum is 1, and the maximum is 2,670.

Thus we can see that, in at least some cases, the splitting procedure happens (notice the drop from the maximum FPI_{obs} measurement to the maximum FPI_{nob} measurement), but that the effect is not large on average. This is a positive sign that the scattering of patches is actually reflective of the underlying concerns, but we cannot be certain from this result. We leave this issue for a qualitative examination in the next section.

4.3 What is the nature of changes that are highly scattered?

Though the majority of patches touched fewer than 5 files, there were a number of more scattered changes. For example, we found that 5.93% of issue reports had patches touching more than 11 files in total. To understand the nature of these scattered changes we conducted a simple qualitative analysis of a random sample of issue reports involving scattered changes.

To select changes for our analysis, we considered all issue reports in Mozilla's issue tracking system associated with the Firefox product, having a status of "Resolved Fixed" or "Verified Fixed". We filtered the resulting list to only include reports that had at least one patch that touched 11 or more files, yielding 943 candidate issue reports. Finally, we randomly selected 50 of those issue reports to use in our analysis. We categorized these 50 as follows.

Arbitrary issue combination. Five of the issue reports in our sample in fact combined multiple issues for the convenience of the developer or reviewers. For example, the patch attached to issue 254982 "includes fixes for bug 249231 and bug 253657" because of "approval backups and waiting checkins." As another example, a developer fixing several related issues (337153, 337830, 339484, 340247, and 340312) created a new issue (340897) and attached patches addressing all of the related issues. Such issue reports would tend to overreport the presence of crosscutting concerns. These are instances of spurious scattering discussed in Section 4.2.

Dead code removal. The Mozilla community places emphasis on maintaining code quality.³ Getting rid of dead/unused code is part of maintaining code quality. But often, it results in removing a lot of files and that ultimately results in an apparently scattered change. We found 9 instances of dead code removal in our sample. In a strict sense, these changes are crosscutting concerns, but they are not particularly significant ones.

New feature addition. Our sample contained 12 issue reports whose patches were scattered because they were

³https://developer.mozilla.org/en/Code_Review_FAQ

new feature requests in Mozilla Firefox. Changes in functionality of the user interface (UI) were more scattered as the same changes are required to be made for multiple platforms simultaneously. For instance, issue 322988 involved the addition a new panel in the browser window for changing bookmark properties. The change was made for two different themes of Firefox, for both Mac OS and Windows. The change was more or less the same for both the themes but resulted in a scattered change. Another instance of a UI related change is issue 353673; in this issue report, a new theme was created for Firefox and it resulted in affecting 245 files in total.

Ripple effect. We observed 4 issue reports in our sample (e.g., 168411, 329741) for which patches are scattered when changes in one part of the codebase affects other parts of the codebase, instances of the familiar ripple effect. Ripple effect is a sign of ineffective modularization, but not necessarily to be improved through the separation of crosscutting concerns.

User-facing changes. Firefox being a web browser greatly depends upon its UI layout, demanding consistency for all its versions. Even minor changes in the layout such as fixing a label in a dialog, making buttons semi-transparent or correcting language errors resulted in a scattered change. We found that even small UI changes can result in a scattered patch partially because different parts of the UI related code are implemented using different programming languages; examples include fixing a label in a dialog, making buttons semi-transparent, and correcting the language of messages displayed to the user. There were 11 such issue reports in our data set. Their associated patches are scattered due to the lack of direct, cross-language support.

Trivial fixes. Some of the changes in our sample were very large but very trivial. For example, changes to the commenting style (issue 338830), fixing punctuation errors (issue 255623), removing white spaces from files (issue 316302) etc. These changes were found to require more review despite their triviality. Because the patch touched many files from different modules, according to Mozilla review standards each module owner is required to review any patch submitted for his module. There were 9 such cases in our data set.

4.4 How scattered are patches?

Assuming that the scattering of patches represents actual crosscutting concerns, we were curious about the distribution of scattering.

Seeing the descriptive statistics given in Section 4.1, it was obvious that we were not dealing with a Gaussian distribution, but that highly scattered concerns could not be dismissed out of hand. We examined histograms of the various measurements to investigate their form; the leftmost portion of the histogram for the FPP measurements (i.e., for FPP of no more than 20) is given in Figure 1. The minimum frequency over this range is 730 for FPP = 20 (i.e., there are 730 patches with an FPP of 20), and the tail of the histogram does not reach 0 for the first time until FPP = 168 after which zeroes occur increasingly often. To visualize this better, we can redraw the histogram on a log-log plot, as per Figure 2. The histogram from the MPP measure has a similar form, as do those from the FPI measures.

The apparent spreading out of the data to the right-hand side of the plot in Figure 2 is due to noise from the

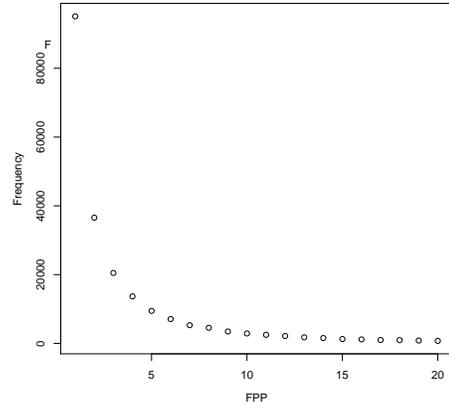


Figure 1: Histogram of FPP (for FPP ≤ 20).

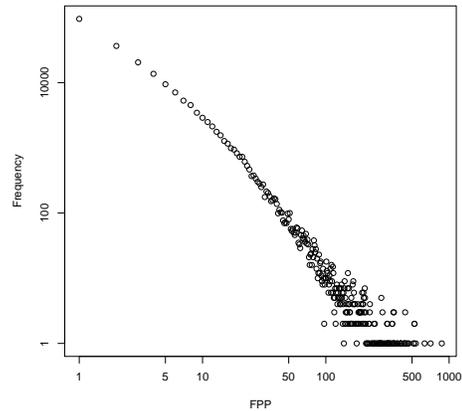


Figure 2: Histogram of FPP (log-log plot).

discrete nature of the histogram; this can be cleaned up by replacing the linearly-scaled bins of the histogram with logarithmically-scaled ones. For example, Figure 3 is the equivalent to Figure 2. The histogram for MPP measurements with logarithmically-scaled bins is shown in Figure 4.

Most importantly, the majority of the data falls at low levels of scattering (79% of patches have 5 or less files affected, 90% have 10 or less; 79% have 1 directory affected, 95% have 3 or less). But a significant number of patches exhibit high or even extremely high degrees of scattering, for both measures.

4.4.1 Heavy-Tailed Distributions

These results exhibit a structure known colloquially as a “heavy tail”, a distribution in which a non-negligible number of events occur at large deviations from the mean. Standard descriptive statistics can give a false impression of the form of these distributions. Such structures have been reported in various natural phenomena, and many studies have examined the presence of heavy-tailed distributions in software; for example, Louridas et al. [26] provide several plots of an appearance similar to Figure 2. Recently, Taube-Schock et

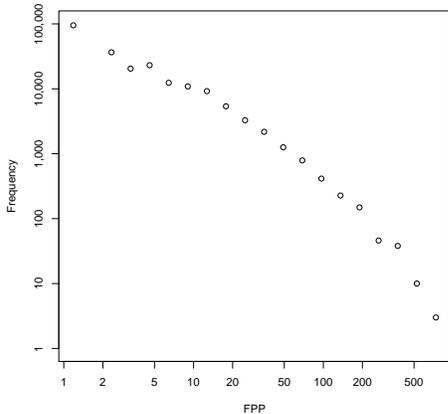


Figure 3: Histogram of FPP (log–log plot; logarithmically-scaled bin widths).

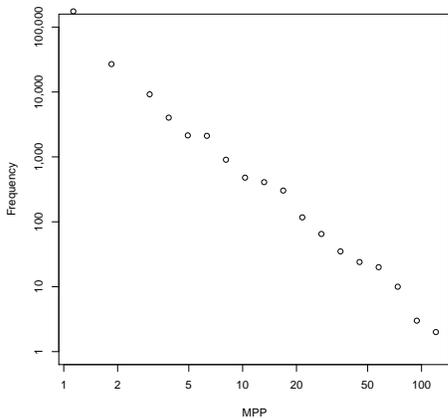


Figure 4: Histogram of MPP (log–log plot; logarithmically-scaled bin widths).

al. [39] reported that every system in a large corpus (of Java systems across domains, at a wide variety of sizes, and differing levels of maturity) exhibited heavy-tailed distributions for coupling, implying the presence of areas of high coupling in every single one of them.

These heavy-tailed distributions are often assumed to follow a *power-law distribution*, the discrete version of which (also known as the *zeta distribution*) is pertinent here. Under this distribution, the probability of an event occurring at some value x is

$$p(x) = C f(x) = \frac{x^{-\alpha}}{\zeta(\alpha, x_{\min})}, \quad (1)$$

where α is a decay constant and C is a normalization constant defined such that $\sum_{x_{\min}}^{\infty} C f(x) dx = 1$. In this case,

$$C = 1/\zeta(\alpha, x_{\min}) = \sum_{n=0}^{\infty} (n + x_{\min})^{-\alpha}$$

where ζ is the generalized or Hurwitz zeta function; x_{\min} is the point above which the power-law holds (as in practice only the tail of the data obeys a power-law).

Finding that the data conforms (or does not) to a power-law distribution is potentially important: certain processes are known to give rise to power laws while other processes give rise to alternative distributions. Thus, being confident about the distribution to which data conforms can both allow us to make good predictions about future data arising from the same process, and to model and thus understand how the data arises in the first place. Falsely claiming conformance to any distribution can be harmful, since it will both lead to false predictions about future data and to misleading conclusions about how the data arises. Thus, questions of data distribution should not be ignored.

The most common procedure for fitting a power-law distribution consists of transforming the data and Equation 1 into a log–log form,

$$\log(p(x)) = -\alpha \log(x) + \log(C), \quad (2)$$

a linear equation in which $-\alpha$ is the slope and $\log(C)$ is the x -intercept. Thus, a linear regression is generally performed to get the best fit straight line; Louridas et al. follow this procedure, for example.

However, as described in detail by Clauset et al. [8], there are a number of serious issues ignored by this procedure: (1) linear regression makes a set of assumptions that are violated in transforming the data into the log–log form; (2) a variety of other distributions also visually follow a linear form in a log–log plot; and (3) it says nothing about how well the power-law distribution model fits the data.

To address each of these issues, Clauset et al. have defined more statistically valid and/or empirically supported alternatives—specifically, numerical procedures for estimating the values of x_{\min} and α ; a set of additional models that can serve as likely alternatives; and a numerical simulation procedure for evaluating the probability that a given model could have generated the observed data. For the details of the fitting procedures and the associated scripts (we used those provided by Clauset for the R statistics language), we refer the reader to the manuscript of Clauset et al. [8] and the companion web page.⁴ The six alternative models suggested as likely alternatives are: the discrete exponential distribution, the discrete log-normal distribution, the discrete power-law distribution with exponential cut-off, the discrete Weibull distribution, the Poisson distribution, and the Yule distribution. We collate some details of the probability estimation procedure in the following subsection.

4.4.2 Estimating goodness-of-fit

Given a set of discrete empirical data $\{x_k\}$, its cumulative distribution function $\hat{P}(x)$ can be directly computed as:

$$\hat{P}(x) = |\{x_k \mid x_k \leq x\}|. \quad (3)$$

Then, any discrete model distribution D with probability mass function $p_D(x)$ will have cumulative distribution function

$$P_D(x) = \Pr(X \leq x) \mid_D = \sum_{k=0}^x p_D(k). \quad (4)$$

To determine goodness-of-fit, we utilize the Monte Carlo simulation procedure of Clauset et al., in which: (1) a synthetic sample is repeatedly drawn (for a predetermined set

⁴<http://www.santafe.edu/~aaronc/powerlaws/>

of repetitions) from the model distribution, (2) a measurement is taken to determine whether the empirical data is a better or worse fit for the model distribution than the synthetic sample, and (3) the value \tilde{p} is computed indicating for what percentage of the synthetic samples the empirical data is a better fit.⁵

To determine whether the empirical data is a better fit to the model distribution than a given synthetic sample, a measure is needed to quantify the distance between two probability distributions. Clauset et al. empirically justify the use of one such measure, the Kolmogorov–Smirnov (KS) statistic:

$$d_{\text{KS}} = \max_{x \geq x_{\min}} |P(x) - P_D(x)| \quad (5)$$

where $P(x)$ is either the cumulative distribution function of the data $\hat{P}(x)$ or of the k th synthetic sample $\widehat{P}_D^k(x)$.

The \tilde{p} value is then computed as

$$\tilde{p} = \frac{\left| \left\{ k \mid 1 \leq k \leq r \wedge d_{\text{KS}}(\hat{P}(x)) \leq d_{\text{KS}}(\widehat{P}_D^k(x)) \right\} \right|}{r}, \quad (6)$$

i.e., the fraction of synthetic samples that are a poorer fit to the model distribution than is the data; r is the total number of synthetic samples to be randomly drawn from the distribution, which Clauset et al. recommend to lie between 1,000 and 10,000. Finally, $\tilde{p} \geq 0.1$ is interpreted as the model distribution being a reasonable fit to the empirical data.

4.4.3 Results of fitting

We followed the Monte Carlo procedure of Clauset et al. to determine whether any of the common candidate distributions are good fits for the data. We arrive at the conclusion that, of the common candidate distributions, the FPP data most closely follows a discrete log-normal distribution ($\mu = -0.26$, $\sigma = 1.69$ for $x_{\min} = 1$) but that even this distribution is not a good fit ($\tilde{p} = 0$ for 1,000 repetitions); selecting slightly higher values for the threshold did not improve the fit. From Figure 3, we can see a slight curve downwards in the data, and so the fact that none of the common candidate distributions model this well is not surprising.

In contrast, the MPP data reasonably fits a discrete log-normal distribution ($\mu = -3.52$, $\sigma = 1.68$ for $x_{\min} = 1$; $\tilde{p} = 0.20$ for 10,000 repetitions).

We then examined the FPI_{nob} data; Figure 5 shows the resulting histogram. This data fits a discrete log-normal distribution ($\mu = -0.69$, $\sigma = 1.74$ for $x_{\min} = 1$) best of the common candidate distributions, although it is a poor fit ($\tilde{p} = 0$ for 1,000 repetitions).

4.4.4 Summary

In conclusion, the data all clearly follows a heavy-tailed distribution, meaning that there are a large number of patches (whether unioned or not over individual issues) with small amounts of scattering, few patches with large amounts of scattering, and very few with very large amounts of scattering. In particular, we can see that scattering follows a continuum: a binary distinction between a crosscutting concern and a non-crosscutting concern seems inappropriate.

None of the data follows a power-law distribution. We can say only that the MPP measurements fit a log-normal

⁵Note that \tilde{p} should not be confused with p used in more common statistical analyses.

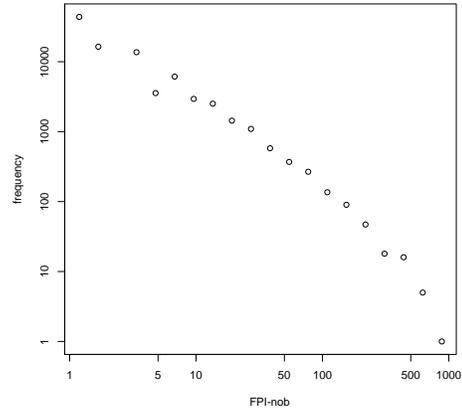


Figure 5: Histogram of FPI_{nob} (log–log plot; logarithmically-scaled bin widths).

distribution, suggesting that it can be modelled well as the superimposition of independent random variables [24] and thus that the entropy-based model of Baldi et al. [4] (see Section 2) has a reasonable basis at the module-level. The FPP and FPI measurements fit none of the standard heavy-tailed distributions, meaning that no standard process models will be a good description for file-level impact of crosscutting concerns. These findings call into question the software engineering literature that follows the common and erroneous fitting process described above to arrive at the conclusion that power laws are present.

More importantly, there are downsides for aspect orientation in these findings. (1) There is no natural cut-off point to differentiate between crosscutting concerns and non-crosscutting concerns, except for the degenerate case when a patch affects only one file. (2) A large number of patches (about half) illustrate at least minor crosscutting; thus, it appears unrealistic to separate and encapsulate all of these, as is promoted by standard AO theory. (3) About half of the patches of Mozilla manifest at least minor crosscutting, so it is a good candidate system on which to test theories about the effects of crosscutting. (4) The processes giving rise to the scattering distribution for the patches must be both complex and poorly understood, meaning that simple model-based solutions are unlikely to suffice.

4.5 Do scattered patches require more revision?

As described in Section 2, crosscutting concerns have been argued as causing defects and it is reported that developers tend to miss the full extent of crosscutting concerns. We would expect one of a few possible consequences to be visible in Mozilla as a result: (1) more highly scattered concerns would require more revision; (2) more highly scattered concerns would more likely become even more scattered after review; (3) the frequency of issue reports would rise over time; (4) the codebase would show signs of decay over time. We address point 1 here, and leave the other points for later.

4.5.1 Scattering versus revision

To measure revision, we used two alternatives: the number of obsolete patches associated with an issue, and the number

of total patches associated with an issue. Neither is a perfect proxy measure, as multiple patches can sometimes be submitted simultaneously, but one would expect that more revisions would generally lead to more (obsolete) patches. We assume that the only reason to have additional patches submitted is either that there is some competition regarding the best solution, or that additional non-obsolete patches signal that the earlier solution was incomplete.

For the number of obsolete patches (per issue) versus the FPI_{nob} measurement, we measured two-sided correlation via both Pearson’s product-moment correlation and Spearman’s rank correlation. Pearson’s indicates a small but positive correlation ($r = 0.21$; $t = 64.26$, $\text{dof} = 93,004$, $p < 2.2 \times 10^{-16}$, 95% confidence = $\{0.20, 0.21\}$). Spearman’s indicates a small but positive correlation ($\rho = 0.34$; $S = 8.87 \times 10^{13}$, $p < 2.2 \times 10^{-16}$).

For the total number of patches (per issue) versus the FPI_{nob} measurement, we again measured two-sided correlation. Pearson’s indicates a marginally larger, positive correlation ($r = 0.29$; $t = 93.45$, $\text{dof} = 93,004$, $p < 2.2 \times 10^{-16}$, 95% confidence = $\{0.29, 0.30\}$). Spearman’s indicates a larger, positive correlation ($\rho = 0.45$; $S = 7.38 \times 10^{13}$, $p < 2.2 \times 10^{-16}$).

Our interpretation is that more highly scattered patches are somewhat more likely to require revision. But is this effect due to confounding correlation between scattering and size of the implementation change?

4.5.2 Scattering versus change size

We considered whether there was a correlation between the scattering measures and the change size measure. For FPP, Pearson’s indicates a middling, positive correlation ($r = 0.48$; $t = 258.03$, $\text{dof} = 221,506$, $p < 2.2 \times 10^{-16}$, 95% confidence = $\{0.48, 0.48\}$); Spearman’s indicates a fairly high correlation ($\rho = 0.72$; $S = 5.03 \times 10^{14}$, $p < 2.2 \times 10^{-16}$). For MPP, Pearson’s indicates a low, positive correlation ($r = 0.19$; $t = 90.82$, $\text{dof} = 221,506$, $p < 2.2 \times 10^{-16}$, 95% confidence = $\{0.19, 0.19\}$); Spearman’s indicates a low-middle, positive correlation ($\rho = 0.37$; $S = 1.14 \times 10^{15}$, $p < 2.2 \times 10^{-16}$).

There are a number of qualitative points that have to be considered. We see that, for FPP, Pearson’s r obtains a middling value while Spearman’s ρ is fairly high. This suggests that there is only a vaguely linear relationship between FPP and change size. Since it should be self-evident that as change size grows beyond the size of individual files, the changes will necessarily become more scattered, and we postulate that this is the effect witnessed. For MPP however, change size is much more loosely constrained by the size of modules, and so we would expect to see a far weaker correlation, which is borne out by the measures.

The third combination, change size versus revision, would require us to merge the individual patch changes in order to measure the complete size. As this is a tricky proposition, we avoid this analysis at present.

4.5.3 Summary

Patches with greater scattering are slightly more likely to be revised, but the confounding effect of larger changes being harder to get right brings into question whether there is a direct effect at work from the presence of crosscutting concerns. Addressing this point will require future work.

ΔFPI	CRs
0	18,198
>0	8,908
<0	7,620

Table 1: Overview of effect of review on scattering.

4.6 Is scattering likely to increase after review?

If there were a correlation between initial scattering and the increase in scattering after review, this would be a signal that more scattered changes are harder for developers to cope with and get right.

We examined the number of revised patches for which the scattering did or did not change. Table 1 overviews the basic observations on changes to the FPI measures for revised patches (i.e., the difference between FPI_{nob} and FPI_{obs} , which we define as ΔFPI): 34,726 issue reports possess obsolete patches (and thus, were revised) out of 93,006 total issue reports addressed, i.e., 37% required revision upon review. The scattering of the majority (52%) of the revised patches was the same after the review (FPI_{nob}) as before (FPI_{obs}).

With respect to ΔFPI , the sets of revised patches for which scattering increased, for which scattering decreased, and for which scattering remained constant, all display a heavy-tailed distribution. Interestingly, for the case where $\Delta FPI < 0$, x_{min} (see Equation 1 and related discussion) is about 12; below this, the distribution is rather flat, suggesting that scattering has to be fairly high before the review process reduces the scattering.

As a second step, we examined whether, for revised patches where the scattering changed, the scattering was more likely to increase than to decrease.

For each issue with both obsolete and non-obsolete patches, we counted the number of files unioned over the obsolete patches and over the non-obsolete patches; we call these counts n_{obs} and n_{nob} respectively. We then constructed a matrix M in which the i, j th cell counted the number of patches for which $n_{\text{obs}} = i$ and $n_{\text{nob}} = j$; the diagonal of M represents patches for which no change in scattering occurred, the upper triangle represents decreased scattering, and the lower triangle increased scattering. The matrix M appeared to be asymmetric, skewed significantly in favour of a *decrease* in scattering.

To test this hypothesis, we performed McNemar’s χ^2 test (not to be confused with Pearson’s) which determines the probability that the marginal frequencies are equal (i.e., it checks whether the matrix deviates from symmetry significantly). The numerical procedure has problems where the input matrix is sparse, so we restricted our attention to the upper left 24×24 submatrix; this captured 32,580 of the issues with revised patches (94% of them) and we treat the remainder as outliers. McNemar’s χ^2 statistic is 782.86, with 276 degrees of freedom, and $p < 2.2 \times 10^{-16}$. This means that the null hypothesis—that M is symmetric—is rejected in favour of the alternative that is asymmetric.

To get the sense of the degree of asymmetry, we restrict our attention to only those issues with both obsolete and non-obsolete patches; we refer to these as the *revised issues*. The mean FPP measure over the revised issues, for *only* obsolete patches, is 7.09; for only non-obsolete patches, it

Year	Files	MB	IRs	IRs/file	IRs/MB
1999	38,512	238	388	0.01	1.96
2000	52,276	311	3,030	0.07	11.03
2001	54,334	358	8,751	0.16	26.16
2002	65,500	400	8,861	0.15	23.38
2003	66,509	361	5,532	0.08	14.54
2004	58,105	329	6,177	0.10	17.90
2005	62,645	353	7,951	0.13	23.32
2006	74,915	476	9,799	0.14	23.64
2007	91,501	508	9,897	0.12	20.12
2008	96,670	533	12,377	0.13	23.78
2009	97,952	555	14,648	0.15	26.93
2010	98,633	567	5,595	0.06	9.97

Table 2: Annual measurements of the codebase and number of issue reports.

is 6.58. Hence, the scattering tends to decrease on revision. By comparison, issues with only non-obsolete patches (i.e., non-revised issues) had a smaller mean FPP (3.27).

4.6.1 Summary

We conclude that review usually does not alter the scattering. In cases when it does, the scattering is slightly more likely to increase than decrease (54% of the time vs. 46%). However, overall, the *degree* of scattering tends to decrease slightly upon revision.

4.7 Is there evidence of decay?

If patches consistently missed some elements of scattered concerns, we would expect to see growth in the number of issue reports as Mozilla increased in bugginess, and hence, its design decayed.

It is clear that the number of issue reports has grown significantly on an annual basis over the lifetime of the Mozilla project, but so has the size of the system. We attempted to download the Mozilla codebase at annual snapshots, specifically, for 1 Jan $\langle year \rangle$ at 00:00:01 Apia time, to represent the total change over the preceding year.⁶ Table 2 shows annual counts of files, millions of bytes, and issue reports, plus issue reports normalized by files and by bytes. Discounting 1999 and 2010 for which only incomplete data was collected, we do not see an obvious, steady growth in normalized counts of issue reports—Mozilla is not obviously getting buggier.

More subtle effects may be at play, however. We examined the proportion of revised patches on an annual basis. There were clearly start-up effects occurring during the years 1999–2001, and the ratio of issues with obsolete patches to the total number of issues was very low (0, 0.002, and 0.11 respectively). After this, a steady-state seems to have been

⁶For the years 1999–2006, this was a straightforward (though slow) checkout from the main branch of the Mozilla CVS repository. In early 2007, Mozilla migrated some (but not all) of its subprojects to a Mercurial repository. It seems that some of the changes in the Mercurial repository migrate to the CVS repository for the sake of supporting the remaining subprojects there, but we are unclear whether this is consistently the case. The distributed nature of Mercurial makes it very difficult to select a “definitive” version of the system for any given timestamp. Thus, we report codebase size as pulled from CVS in all cases, for practicality.

reached: the mean ratio for the remaining years is 0.42 (with a standard deviation of 0.02).

To determine whether the variations are explainable by random chance, we analyzed the data for non-obsolete patches on an annual basis and ignored the years 1999–2001 due to the start-up effects. For each of the remaining years, with respect to the FPI_{nob} measure, we collated the probability mass function $\hat{p}(x)$ for the data from that year. We then calculated an average probability mass function $\tilde{p}(x)$ across the years 2002–2010 by taking the mean across all these years for each discrete value of x and normalizing the result; $\tilde{p}(x)$ served as a model distribution against which to compare the annual data. We then followed the Monte Carlo procedure of Clauset et al. (Section 4.4.2) to determine whether each year’s data fit $\tilde{p}(x)$ better than a synthetic sample randomly drawn from $\tilde{p}(x)$.

For each set of annual data for 2002–2010, we generated 1,000 synthetic samples from $\tilde{p}(x)$ and compared their KS statistic to that of the empirical data. The resultant \tilde{p} value was 0 in all cases—a very strong indication that the variations in the data *cannot* be explained by random chance.

We then examined the data for trends in the annual variations, shown in Table 3. After initial startup effects, we see a clear trend towards higher scattering, as the mean level of FPI_{nob} increases steadily though slowly from 2005 through 2010. But we see that the number of issues dealt with annually increases rapidly: this may either be a sign that the system’s design is decaying (despite the lack of a strong signal in Table 2) or that its size (and hence, feature set) is expanding. That a bigger, more complex system has slightly higher levels of scattering is not surprising.

In order to assess the annual variations, we needed a means to evaluate the overall annual scatteredness that is independent from the number of issues addressed. Consider again that if the exact same distribution fit each year’s data, clearly, there would be no change in the (normalized) scatteredness of patches, even if the mean (non-normalized) scattering were changing. Since a probability mass function is necessarily normalized, the number of data points involved is irrelevant (the figures earlier in the paper were not shown with normalized axes) and hence we would expect that the slope would remain constant. On the other hand, if the scatteredness is increasing but we hold the number of data points fixed, this will have the effect of spreading the data to the right, and decreasing the height of the initial peak. For simplicity, we fit the power-law distribution to each annual set of data to suggest its shape. The slope of this data (which is the negative of the α parameter, as per Equation 2) can be compared across years to determine how the scatteredness is changing— α serves as a relative measure of scatteredness over whole systems. The fact that a power-law distribution does not fit a data set well is potentially problematic, since the α parameter will then not be a true representation of the scatteredness in a statistically sound sense, but it will suffice here to indicate trends.

Examining the α parameter for each year’s best fitted power law, we see a slow decrease from 2005 to 2010. A lower α indicates an increase in issue reports with high FPI_{nob} measurements relative to issue reports with low FPI_{nob} measurements (i.e., the long-tail is stretching): scattering is getting slowly more pronounced, *independently* of the growth in the number of issue reports handled annually. This means that Mozilla shows *minor* signs of decay.

Year	$\mu(\text{FPI}_{\text{nob}})$	α	Year	$\mu(\text{FPI}_{\text{nob}})$	α
1999	2.29	2.17	2005	3.89	1.89
2000	3.14	1.97	2006	4.19	1.86
2001	5.13	1.81	2007	4.46	1.79
2002	4.18	1.87	2008	4.87	1.76
2003	4.34	1.74	2009	4.93	1.73
2004	4.18	1.86	2010	5.11	1.72

Table 3: Change trends in the annual FPI_{nob} data.

5. THREATS TO VALIDITY

In the Mozilla issue tracking system, developers sometimes submit patches for more than the issue supposedly being addressed. Ideally, the project’s patch review process would catch this, but it is not likely to be perfect. Nevertheless, this phenomenon would have the effect of artificially *increasing* the scattering of patches, but the impact of scattering is already low, so this effect can be ignored.

On the other hand, a patch may fail to fully address an issue, and hence underrepresent the appropriate level of scattering. One would presume that such cases would be caught when a patch failed to completely address an issue, or else we would see a steady rise in issue reports independent from the trends in system size.

Also, the Mozilla codebase is designed for multiple platforms and this results in patches containing files that are written in multiple programming languages. Most of the code is written in C++, but there are Javascript and XML files to support cross-platform compatibility in a clone-like manner. (We note that such polylingual crosscutting concerns are not refactorable to aspects with current techniques.) Cross-platform support will tend to raise the levels of scattering, so the observed scattering in the system is again potentially worse than a monolingual system.

As the system considered in this study is open source, there are many possible sources of defects including the complexity of problem domain. The developers’ experience likely varies widely, with contributions arriving daily from around the world. We expect that developer’s experience with the codebase and the programming language used will have an impact on the amount of scattering observed.

Our study does not consider factors of developer effort in coping with more scattered concerns. If scattered concerns require a disproportionate effort to deal with, we would expect that developers would do a poorer job with them, leading to more revision or more issues being reported. Neither effect is observed.

6. CONCLUSION

We have analyzed the submitted patches for the Mozilla project over a 10-year period, examining their scattering properties to consider whether scattering causes evolutionary problems. We found that real crosscutting concerns occur in Mozilla, but that scattering of its patches forms a continuum: patches are not simply either scattered or not. Thus, there is no natural distinction to be made between crosscutting concerns and non-crosscutting concerns—roughly half of the submitted patches show at least some scattering.

Furthermore, while our measures of scattering take the form of a heavy-tailed distribution, they fail statistical tests

to be a power-law distribution or any other typical candidate distributions (the MPP measure is an exception). This implies that we have no good process models at our disposal to describe how scattering arises at the file-level in Mozilla, and hence simple model-based approaches are unlikely to easily capture the nuances. This also contradicts the common assumption in the software engineering literature that heavy-tailed distributions imply power laws.

Patches with greater scattering are slightly more likely to be revised upon review than not, but the degree of scattering does not change in the majority of cases upon revision; when it does, the degree of scattering tends to *decrease* slightly on average. This contradicts the prediction derived from aspect oriented theory that elements in a scattered concern are more likely to be overlooked, suggesting that modifications to some elements would likely be added based on review.

While the absolute number of change requests has shown steady growth for Mozilla, so has the size of the codebase: normalized counts of issue reports show no obvious growth pattern suggestive of increasing bugginess. On the other hand, there is a subtle annual increase of scatteredness of patches, independent of the growth in issue reports. This suggests that Mozilla’s structure is getting slowly more brittle, but the effect is small. Overall, this contradicts the standard aspect-oriented theory that any crosscutting concern will lead to significant evolutionary problems, and hence that refactoring to aspect-oriented constructs is warranted.

One issue that our study has not addressed is whether particular kinds of crosscutting concerns cause problems, as some of the literature has suggested. If such crosscutting concerns are in the minority within Mozilla, their effects may be washed out in our analyses. Future work will need to attempt to distinguish different kinds of crosscutting concerns within a system like Mozilla to examine if these are more consistently or severely problematic.

The case for aspect orientation rests on the assumption that crosscutting concerns will cause significant evolutionary problems. Our results on Mozilla appear to contradict this assumption. Of course, Mozilla is only a single system—a single, open-source, massive, heavily used, and long-lived system containing a wide variety of features commonly pointed to as archetypes for crosscutting concerns. If the results were not to generalize, there would have to be something special and important about Mozilla or the processes used in its development, warranting close study as a potential alternative to the explicit separation of crosscutting concerns.

7. ACKNOWLEDGMENTS

We wish to thank Brad Cossette, Rylan Cottrell, and Soha Makady for their insightful comments on this work. This work was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada.

8. REFERENCES

- [1] J. Asundi and R. Jayant. Patch review processes in open source software development communities. In *Proc. Hawaii Int. Conf. Syst. Sci.*, pp. 166c/1–166c/7, 2007.
- [2] L. Aversano et al. Relationship between design patterns defects and crosscutting concern scattering degree. *IET Softw.*, 3(5):395–409, 2009.

- [3] K. Ayari et al. Threats on building models from CVS and Bugzilla repositories. In *Proc. IBM Conf. Centre Adv. Stud. Collab. Res.*, pp. 215–228, 2007.
- [4] P. Baldi et al. A theory of aspects as latent topics. In *Proc. ACM SIGPLAN Conf. Object-Oriented Progr. Syst. Lang. Appl.*, pp. 543–562, 2008.
- [5] C. Baldwin and K. Clark. *Design Rules: The Power of Modularity*, volume 1. MIT Press, 1999.
- [6] M. Bartsch and R. Harrison. An exploratory study of the effect of aspect-oriented programming on maintainability. *Softw. Qual. J.*, 16(1):23–44, 2008.
- [7] S. Breu et al. Investigating information needs to improve cooperation between developers and bug reporters. In *Proc. Conf. Comput. Supported Coop. Work*, pp. 301–310, 2010.
- [8] A. Clauset et al. Power-law distributions in empirical data. *SIAM Rev.*, 51(4):661–703, 2009.
- [9] M. Eaddy et al. Do crosscutting concerns cause defects? *IEEE Trans. Softw. Eng.*, 34(4):497–515, 2008.
- [10] E. Figueiredo et al. Evolving software product lines with aspects. In *Proc. Int. Conf. Softw. Eng.*, pp. 261–270, 2008.
- [11] E. Figueiredo et al. Crosscutting patterns and design stability. In *Proc. Int. Conf. Program Comprehen.*, pp. 138–147, 2009.
- [12] E. Figueiredo et al. On the impact of crosscutting concern projection on code measurement. In *Proc. Int. Conf. Aspect-oriented Softw. Dev.*, pp. 81–92, 2011.
- [13] F. Filho et al. Exceptions and aspects: The devil is in the details. In *Proc. ACM SIGSOFT Int. Symp. Foundations Softw. Eng.*, pp. 152–162, 2006.
- [14] A. Garcia et al. Modularizing design patterns with aspects. In *Proc. Int. Conf. Aspect-Oriented Softw. Dev.*, pp. 3–14, 2005.
- [15] D. German. Automating the measurement of open source projects. In *Proc. Wkshp. Open Source Softw. Eng.*, pp. 63–67, 2003.
- [16] D. German. An empirical study of fine-grained software modifications. In *Proc. IEEE Int. Conf. Softw. Maintenance*, pp. 316–325, 2004.
- [17] P. Greenwood et al. On the impact of aspectual decompositions on design stability. In *Proc. Europ. Conf. Obj.-Oriented Progr.*, pp. 176–200, 2007.
- [18] A. Ihara et al. An analysis method for improving a bug modification process in open source software development. In *Proc. ACM Int. Wkshp. Princip. Softw. Evol./ERCIM Wkshp. Softw. Evol.*, pp. 135–144, 2009.
- [19] D. Kelly. A study of design characteristics in evolving software using stability as a criterion. *IEEE Trans. Softw. Eng.*, 32(5):315–329, 2006.
- [20] G. Kiczales et al. Aspect-oriented programming. In *Proc. Europ. Conf. Object-Oriented Progr.*, pp. 220–242, 1997.
- [21] J. Kienzle and S. G elineau. AO challenge: Implementing the ACID properties for transactional objects. In *Proc. Int. Conf. Aspect-Oriented Softw. Devel.*, pp. 202–213, 2006.
- [22] B. Kitchenham et al. Misleading metrics and unsound analyses. *IEEE Softw.*, 24(2):73–78, 2007.
- [23] M. LaMantia et al. Analyzing the evolution of large-scale software systems using design structure matrices and design rule theory. In *Proc. IEEE/IFIP Working Conf. Softw. Arch.*, pp. 83–92, 2008.
- [24] E. Limpert et al. Log-normal distributions across the sciences. *BioScience*, 51(5):341–352, 2001.
- [25] R. Lopez-Herrejon and S. Apel. Measuring and characterizing crosscutting in aspect-based programs. In *Proc. Int. Conf. Fundamental Approaches Softw. Eng.*, pp. 423–437, 2007.
- [26] P. Louridas et al. Power laws in software. *ACM Trans. Softw. Eng. Methodol.*, 18(1):1–26, 2008.
- [27] A. MacCormack et al. Exploring the structure of complex software designs. *Manage. Sci.*, 52(7):1015–1030, 2006.
- [28] M. Marin et al. A classification of crosscutting concerns. In *Proc. IEEE Int. Conf. Softw. Mainten.*, pp. 673–676, 2005.
- [29] M. Marin et al. Documenting typical crosscutting concerns. In *Proc. Working Conf. Reverse Eng.*, pp. 31–40, 2007.
- [30] A. Mockus et al. Two case studies of open source software development. *ACM Trans. Softw. Eng. Methodol.*, 11(3):309–346, 2002.
- [31] M. Nurolahzade et al. The role of patch review in software evolution. In *Proc. ACM Int. Wkshp. Princip. Softw. Evol./ERCIM Wkshp. Softw. Evol.*, pp. 9–18, 2009.
- [32] D. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, 1972.
- [33] A. Przybytek. An empirical assessment of the impact of aspect-oriented programming on software modularity. In *Proc. Int. Conf. Eval. Novel Approaches Softw. Eng.*, pp. 139–148, 2010.
- [34] A. Przybytek. Where the truth lies: AOP and its impact on software modularity. In *Proc. Int. Conf. Fundamental Approaches Softw. Eng.*, pp. 447–461, 2011.
- [35] M. Revelle et al. Understanding concerns in software. In *Proc. IEEE Int. Wkshp. Progr. Comprehen.*, pp. 23–32, 2005.
- [36] J. Siadat et al. Optimization aspects in network simulation. In *Proc. Int. Conf. Aspect-Oriented Softw. Dev.*, pp. 122–133, 2006.
- [37] K. Sullivan et al. The structure and value of modularity in software design. In *Proc. Europ. Softw. Eng. Conf./ACM SIGSOFT Int. Symp. Foundations Softw. Eng.*, pp. 99–108, 2001.
- [38] P. Tarr et al. N degrees of separation: Multi-dimensional separation of concerns. In *Proc. Int. Conf. Softw. Eng.*, pp. 107–119, 1999.
- [39] C. Taube-Schock et al. Can we avoid high coupling? In *Proc. Europ. Conf. Object-Oriented Progr.*, pp. 204–228, 2011.
- [40] S. Yau and J. Collofello. Design stability measures for software maintenance. *IEEE Trans. Softw. Eng.*, 11(9):849–856, 1985.
- [41] C. Zhang and H.-A. Jacobsen. Quantifying aspects in middleware platforms. In *Proc. Int. Conf. Aspect-Oriented Softw. Dev.*, pp. 130–139, 2003.