Does Feature Scattering Follow Power-Law Distributions? An Investigation of Five Pre-Processor-Based Systems

Rodrigo Queiroz
Federal University of Minas Gerais, Brazil

Leonardo Passos
University of Waterloo
Canada

Marco Tulio Valente
Federal University of Minas Gerais, Brazil

Sven Apel
University of Passau
Germany

Krzysztof Czarnecki
University of Waterloo
Canada

ABSTRACT

Feature scattering is long said to be an undesirable characteristic in source code. Since scattered features introduce extensions across the code base, their maintenance requires analyzing and changing different locations in code, possibly causing ripple effects. Despite this fact, scattering often occurs in practice, either due to limitations in existing programming languages (e.g., imposition of a dominant decomposition) or time-pressure issues. In the latter case, scattering provides a simple way to support new capabilities, avoiding the upfront investment of creating modules and interfaces (when possible). Hence, we argue that scattering is not necessarily bad, provided it is kept within certain limits, or thresholds. Extracting thresholds, however, is not a trivial task. For instance, research shows that some source-code-metric distributions are heavy-tailed, usually following power-law models. In the face of heavy-tailed distributions, reporting metrics in terms of averages and standard deviations is unreliable, although commonly done so. Thus, prior to extracting reliable thresholds for feature scattering, one must understand the shape of feature-scattering distribution. In this direction, we analyze the scattering degree of five C-pre-processor-based software families and verify whether their empirical cumulative feature-scattering distributions follow power laws. Our results show that feature scattering in the studied subject systems have characteristics of heavy-tailed distributions, with a good-fit with power laws. Hence, we raise awareness that feature scattering thresholds based on central measures may not be reliable in practice.

Categories and Subject Descriptors

D.2.8 [**Metrics**]: Product metrics

FOSD '14 September 14 2014, Västerås, Sweden

http://dx.doi.org/10.1145/2660190.2662114.

General Terms

Measurement, Reliability, Verification

Keywords

Software families, pre-processor, feature orientation, scattering, power-law distribution

1. INTRODUCTION

Feature scattering is long said to be an undesirable property of source code [2, 3, 9, 14, 15, 22]. Since scattered features introduce extensions across the code base, their maintenance requires changing different locations in code, possibly causing ripple effects. Despite this fact, scattering often occurs in practice [9, 14, 15, 16, 17, 22], either due to limitations in existing programming languages (e.g., imposition of a dominant decomposition [23, 13]) or time-pressure issues. In the latter case, scattering provides a simple way to support new capabilities, avoiding the upfront investment of creating modules and interfaces (when possible). Hence, we argue that scattering is not necessarily bad, provided it is kept within certain limits, or *thresholds*.

Different from other metrics [1, 20], reference thresholds for feature scattering have not been defined so far. At best, researchers report averages and standard deviations over a large set of systems, as performed by Liebig et al. [16]. Central tendency measures (e.g., mean and standard deviation), however, might not be representative values. Recent work [4, 18, 20, 24], for instance, suggests that some source-code metrics follow *heavy-tailed* distributions, often times matching a power-law distribution. In such distribution kind, the probability that an entity measure deviates from a typical value (e.g., mean) is not negligible. Stated otherwise, a significant fraction of code entities do not follow typical metric values, making central statistics unreliable [4].

As a first step towards extracting reliable thresholds for feature scattering, we set to study the shape of feature-scattering distributions in real-world systems. Specifically, we concentrate in systems that rely on the C pre-processor as a means to represent their features, and analyze five open-source software families matching that criterion.

The C pre-processor is largely used to extend the C language with simple meta-programming facilities, in addition to supporting the implementation of software families [16, 2, 21]. In the latter case, features are denoted with macro

elfgcchack.h

```
...
4494  #if defined(LIBXML_FTP_ENABLED)
...
4497  extern __typeof (xmlIOFTPClose) xmlIOFTPClose \
 __attribute((
    alias("xmlIOFTPClose__internal_alias"))
 );
4498  #else
...
4500  extern __typeof (xmlIOFTPClose) \
  xmlIOFTPClose__internal_alias \
 __attribute((visibility("hidden")));
...
4502  #endif
...
```

Figure 1: Feature example (libxml2)

names, which in turn are referenced by different compilation guard conditions. In the C pre-processor, there are different types of guard conditions: `#ifdef`, `#ifndef`, `#elif`, and `#if`. In this paper, we refer to all these constructs simply as *ifdefs*.

Figure 1 provides an example taken from libxml2, one of our subject systems. In file `elfgcchack.h`, developers introduce some code fragments conditioned to the presence of specific features. Lines 4,495–4,497, for instance, depend on the presence of feature `LIBXML_FTP_ENABLED`. Selecting this feature causes the C pre-processor to keep the definition of `xmlIOFTPClose` as given in line 4,497; otherwise, the absence of feature `LIBXML_FTP_ENABLED` results in the alternative fragment in line 4,500. To select a feature, users/developers define them in code, as in

```
#define LIBXML_FTP_ENABLED
```

or set specific compiler flags.[1]

We analyze scattering by parsing *ifdefs* in the code of five C-pre-processor-based open-source software families: vi, libxml2, lighttpd, MySQL, and the Linux kernel. Specifically, for each system, we measure the scattering degree of each of its features (i.e., the number of *ifdefs* that refer to each feature), checking whether the resulting empirical cumulative distribution function (CDF) follows a power law.

In a nutshell, our investigation suggests that feature scattering does follow a power-law distribution. Thus, we raise awareness that feature-scattering thresholds based on central measures may not be reliable in practice. The investigation of a larger corpus, however, is required to further confirm (or refute) our findings.

The remainder of this paper is organized as follows: In Section 2, we provide a short explanation of power-law distributions, which we reference throughout the paper. Section 3 presents our methodology when investigating the relationship between scattering and power laws. We then discuss our results in Section 4, followed by a discussion of threats to validity (Section 5) and related work (Section 6). Section 7 then concludes the paper and points out future work.

[1]Example: `-DLIBXML_FTP_ENABLED` in gcc.

2. POWER-LAW DISTRIBUTIONS

A discrete power-law distribution (which we simply refer as power law) is a probability distribution where the probability that a discrete random variable X assumes a value x is proportional to x raised to the negative power of a positive constant k:

$$P(X = x) \propto cx^{-k} \qquad where \quad c > 0, k > 0 \tag{1}$$

A power law as given by the equation diverges when $x = 0$. In fact, it requires a lower-bound value $x_{min} > 0$ to set up a cut-off value defining a starting point ($x > x_{min}$) from which a power-law behaviour occurs [5]. As we shall see later in the methodology part of our work, the parameters k and x_{min} play an important role when performing a goodness-of-fit analysis.

An important characteristic of a power-law distribution concerns its plotting: When plotted in a logarithmic scale, a power-law function results in a decreasing line. As Baxter et al. [4] report, this is a distinguishing characteristic of power-law distributions.

Interestingly, different researchers report that the distribution of different source-code metrics follow a power-law distribution [24, 4, 18, 20]. Currently, however, no work has investigated whether the same behaviour occurs for feature scattering—our goal of investigation.

3. METHODOLOGY

This section discusses the selection of subject systems, the data collection process to assess feature scattering, and the statistical analysis of the collected data. Our datasets, R scripts, and associated tooling are publicly available on a companion website.[2]

3.1 Selection of Subjects

To verify whether scattered features follow a power-law distribution, we selected five open-source software families that rely on C pre-processor directives to annotate feature code. Table 1 provides information of the subjects.

Our selection takes a subset of the systems analyzed by Liebig et. al [16] when investigating 40 open-source systems. The choice of our subjects was guided by three criteria: (i) each system must cover a distinct functional domain. This criterion prevents bias towards an specific domain; (ii) each system must be a mature software system in the selected functional domain. This decision relies on the assumption that developers of mature systems are more likely to have found a practical balance to when and how much to scatter. It is a requirement for establishing credible thresholds; (iii) the selection of subjects must comprise systems with different sizes, which we measure using two distinct metrics (avoids bias towards a particular system size):

- SLOC (Source Lines of Code): The total number of source lines of code of a given system. This counting excludes blank lines and comments. Moreover, sequence of multilines (lines ending with a backslash) are counted as a single line.[3]

[2]`http://rodrigoqueiroz.bitbucket.org/fosd2014.html`

[3]Multilines are convenient when spanning a long line across multiple ones; during compilation, sequences of multilines are taken as a single line.

Table 1: Selected subjects (general description)

System	Version	Source Code Location	Functional Domain
vi	50325	`http://sourceforge.net/projects/ex-vi`	Text editor
libxml2	2.9.1	`https://git.gnome.org/browse/libxml2`	XML library
lighttpd	1.4.35	`http://www.lighttpd.net/download`	Web server
MySQL	5.6.19	`http://dev.mysql.com`	Database management system
Linux kernel	3.15	`https://github.com/torvalds/linux`	Operating system

Table 2: Selected subjects: size metrics (SLOC: Source Lines of Code, NOFC: Number of Feature Constants)

System	SLOC	NOFC
vi	22,275	118
libxml2	222,009	2,095
lighttpd	39,990	179
MySQL	1,578,250	1,987
Linux kernel	11,964,252	12,653

- NOFC (Number of Feature Constants): The total number of macros names that are referred in, at least, one *ifdef*.

As shown in Table 2, our subjects include small (vi and lighttpd), medium (libxml2), and large software systems (MySQL and Linux kernel), as quantified by the SLOC and NOFC metrics. All subjects are mature systems, and have existed over many years.

3.2 Data Collection

To assess feature scattering, we first parse the source files (implementation and header C files) of each target system. Parsing is performed using the src2srcml tool,[4] which outputs an XML representation of the code. The resulting XML files preserve all the code, including macro directives (src2srcml does not perform any pre-processing).

With all annotations in place, we run a custom-made tool (fscat) to process the XML files produced by src2srcml and measure the scattering degree of each feature.

The scattering metric reported by fscat defines the scattering of a feature as the number of *ifdefs* that refer to it. In the previous example of Figure 1, for instance, the scattering degree of `LIBXML_FTP_ENABLED` is one for the presented code fragment.

3.3 Statistical Analysis

Our statistical analysis is performed using the R statistical environment[5] and the poweRlaw package [11].

First, we define the two parameters k and x_{min} of the best-fit power law ($\mathbb{P}_0$) that approximates as close as possible to the empirical CDF of the scattering distribution ($\mathbb{P}$) of a system under analysis. When searching for a $\mathbb{P}_0$, we rely on the maximum-likelihood estimator method (MLE), while choosing x_{min} as the value minimizing the Kolmogorov-Statistic (KS). The KS statistic is given by the maximum distance $|\mathbb{P}_0(x) - \mathbb{P}(x)|$, for all $x > x_{min}$. For further details, the reader is referred to elsewhere [5, 12, 11].

[4] `http://www.sdml.info/projects/srcml/`
[5] `http://www.r-project.org/`

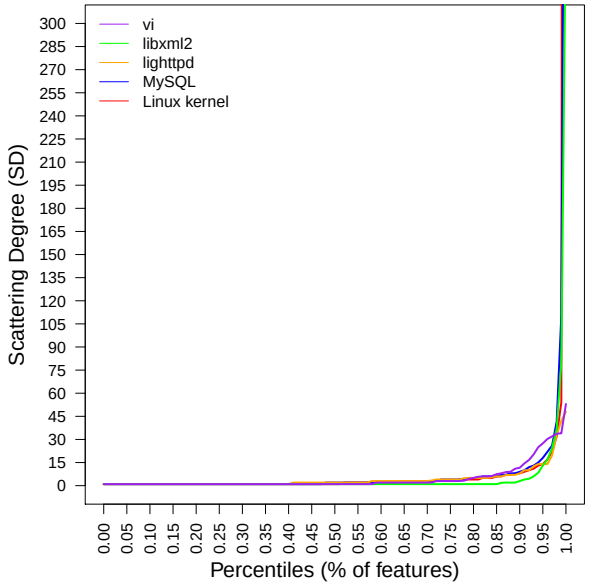


Figure 2: Cropped percentile-plot of the scattering degree

Once we estimate k and x_{min}, we perform a hypothesis test to verify whether a power-law distribution is a plausible model for the behaviour of each empirical CDF. Following Clauset et al. [5], we perform a goodness-of-fit test via a boostrap procedure following the algorithms and steps outlined in [12]. Simply put, we generate 2,500 datasets from the $\mathbb{P}_0$ model and then try to re-infer a new best-fit power law for each generated dataset. The *p-value* of the simulation process corresponds to the fraction of times the KS of the best-fit model of the generated dataset is higher than the one obtained for $\mathbb{P}$. Our hypotheses are the following:

- Null hypothesis: $\mathbb{P}_0$ is a plausible fit for $\mathbb{P}$
- Alternative hypothesis: $\mathbb{P}_0$ is not a plausible fit for $\mathbb{P}$

As Clauset et al. argue, if the test reports a *p-value* larger than 0.1, one must accept the null hypothesis. Otherwise, it should be rejected in favour of the alternative hypothesis. In the latter case, $\mathbb{P}$ is unlikely to conform to a power-law distribution; rather, the empirical CDF function is better fit in another model, and may or not be heavy-tailed.

4. RESULTS

To assess distribution of the feature-scattering degrees, we plot the percentile graph of the empirical CDF of each

Table 3: Max and mean scattering degree (SD) value

System	Max SD	Mean SD
vi	53	4.73
libxml2	379	4.15
lighttpd	48	4.21
MySQL	652	6.99
Linux kernel	2,698	5.43

subject system. As Figure 2 shows, the highest scattering-degree (SD) value in 95% of the features in each system ranges from 13 to 27. In contrast, the remaining 5% of features have a steep scattering degree (due to space, we crop the y-axis of the plotted graph to report a maximum SD of 300). In the latter share, the scattering degree reaches extreme values in MySQL and in the Linux kernel, while other systems display lower bounds (vi, libxml2, and lighttpd). Table 3 shows the maximum SD in each system, along with its mean value.

The analysis of the percentile plot shows that the mean SD-value is too far apart from the values in the last 10% of the features. This nicely illustrates that the mean is not a representative value of the scattering degree of a typical feature. Such characteristic is a necessary, yet not sufficient condition for us to claim that feature scattering follows a power-law distribution in the analyzed systems. To this end, we perform a best-fit analysis.

Following the methodology steps described in Section 3.3, we perform a best-fit analysis to estimate the parameters of the power-law model (k and x_{min}) of each empirical CDF. In Figure 3, we plot each power law in logarithmic scale, resulting in the red decreasing line in each graph of the figure. As we state in Section 2, the resulting line is a distinct characteristic of power-law distributions. We then take the logarithmic scale of the scattering measures of the empirical CDF of each system, adding the transformed data points to the graph of each system. The resulting plot reveals that the data points approximate the line of the power law, strengthening our understanding that scattering follows a power-law distribution.

To statistically check whether the fitted power laws are plausible models (null hypothesis), we perform a bootstrapping hypothesis test. Table 4 shows the *p-values* obtained for each power law. We find four statistically significant models ($p\text{-}value > 0.1$), leading us to accept the null hypothesis for vi, lighttpd, MySQL, and Linux kernel. In the case of libxml2, however, we reject the null hypothesis in favour of the alternative one (the power-law model is not a plausible explanation model). Note that this is not the same as concluding that the scattering distribution of libxml2 is not heavy-tailed. In fact, Figure 2 suggests a heavy-tailed behaviour. Thus, feature scattering in libxml2 is likely to better fit an alternative heavy-tailed model (e.g., stretched exponential or log-normal).

Summary of findings: Feature-scattering distributions follow power laws in four out of five systems we analyze. In the remaining system (libxml2), feature scattering seems to follow an alternative heavy-tailed distribution.

5. THREATS TO VALIDITY

The most likely threat to the validity of our findings is the selection of only five subject systems (external validity). We acknowledge that the current selection is not large enough for us to judge whether our results are applicable to every pre-processor-based software family. We, however, attempted to minimize this threat by carefully selecting mature systems of different sizes and that come from different domains. The mechanism in which features are realized also poses an external threat. Since features may be realized in different ways depending on the programming language, scattering may not have the same behaviour as observed in the five investigated C-pre-processor-based software families.

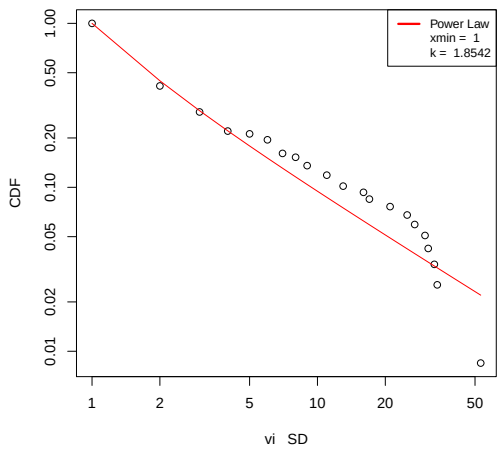
Another threat arises when computing the scattering degree. When using fscat, we consider all the C source of each system, and we do not distinguish files that are automatically generated (e.g., those produced by parser generators) from those that are not. Thus, our results are, to some extent, subject to the influence of such particular files (construct validity). We argue, however, that the majority of files we take for analysis are not automatically generated.

Last, but not least, our results indicate that feature scattering follows a power-law distribution in four out of five of our subjects. However, it might be the case that other distributions are a better fit (e.g., log-normal or stretched exponential). We defer such an investigation for future work.

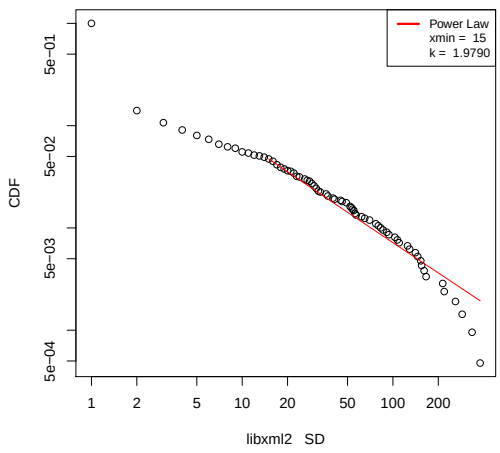
6. RELATED WORK

Liebig et al. [16] analyzed forty program families to show how developers employ the C pre-preprocessor when implementing features and their associated *ifdefs* in code. The authors consider not only scattering metrics, but also metrics measuring tangling (the degree in which features interplay in *ifdef* annotations), the granularity of annotations (the syntactic location where an *ifdef* occurs—e.g., at a global level, inside a function, inside a block, etc.), and the type of annotated code (homogeneous, meaning that a verbatim copy of the annotated also appears in another annotated code; heterogeneous, with distinct extensions; or a mix of the previous two). The authors report their results using central statistics measures, including averages and standard deviations. However, the properties of the underlying metric-distributions have not been checked (e.g., whether data is symmetric, as seen in Gaussian distributions, or whether they are heavy-tailed, as in power-law distributions), which may turn results not representative of true typical values. Moreover, the authors performed transformations in the annotations of their investigated subjects (e.g., by propagating the condition of outer *ifdefs* to inner ones and making a conjunction of each *elif* condition with the condition of preceding branches.) We argue that such transformations may lead to over-scattering. In contrast, we measure scattering in a conservative manner, taking it as is given explicitly in the source code.

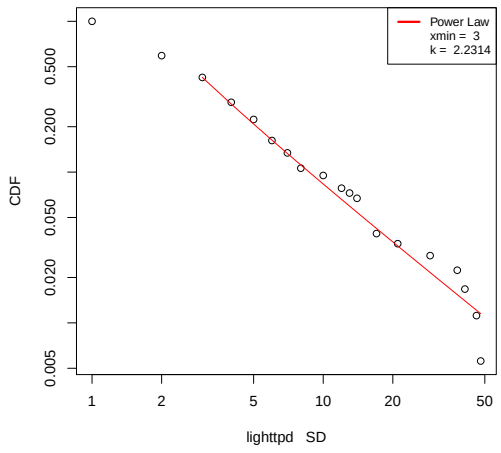
Couto et al. [7] extracted a software product line from the ArgoUML tool, using *ifdefs* to annotate feature code. As part of their study, the authors report similar metrics to those used by Liebig et al., including the scattering degree of features. With the observed scattering values, the authors link the corresponding features to specific patterns reported by Figueiredo et al. [10]. The patterns reported by



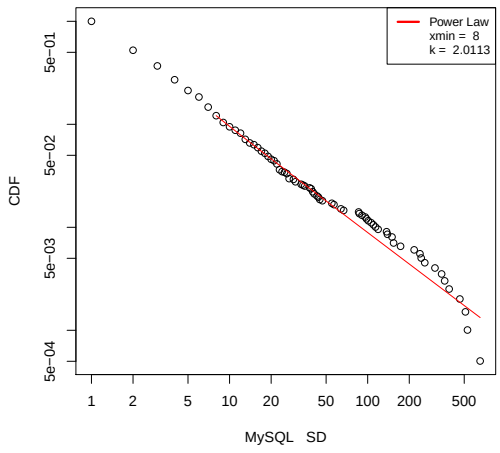
(a)



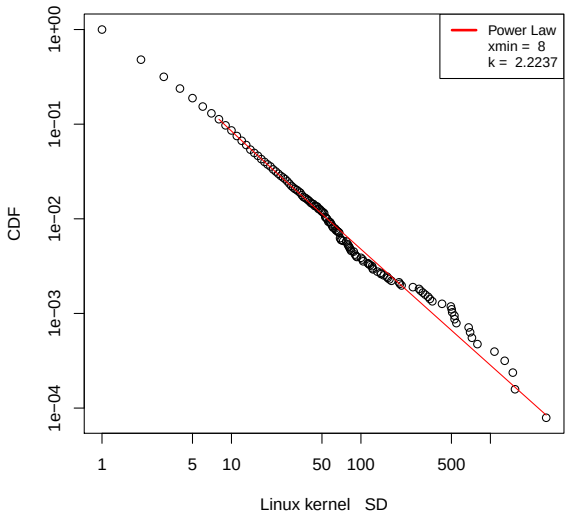
(b)



(c)



(d)



(e)

Figure 3: Power-law distribution of the scattering degree (SD) values in logarithmic scale

Table 4: Power-law inferred parameters and their statistical significance

System	k	xmin	p-value	Statistically significant?
vi	1.8542	1	0.2688	Yes
libxml2	1.9790	15	0.0756	No
lighttpd	2.2314	3	0.8752	Yes
MySQL	2.0113	8	0.2368	Yes
Linux kernel	2.2237	8	0.6436	Yes

Figueiredo et al. formalize rules on how to identify specific kinds of scattered features.

Other researchers [8] investigated the relation between scattering and bugs, but do not prescribe a threshold limiting the degree of scattering. Nonetheless, they provide evidence that simple metrics as the scattering degree (a.k.a. *concern diffusion metric*) correlate with the number of bugs in a system, independent of the size of the latter. This strengthens our claim towards the need of reliable thresholds.

Outside the feature-oriented and product-line community, there are different pieces of work checking the characteristic distribution of size, coupling, and cohesion-related metrics. For example, Louridas et al. [18] study the existence of power-law distributions in different kinds of software components, including Java classes, Perl packages, shared Unix Libraries, and Windows dynamic linked libraries (DLLs). The authors conclude that heavy-tailed distributions, usually power-law distributions, appear at various levels of abstraction, in many domains, operating systems, and languages. Concas et al. [6] study ten different properties related to classes and methods of a large Smalltalk system, consistently finding non-Gaussian distributions for these properties. The authors then conclude that "*the usual evaluation of systems based on mean and standard deviation of metrics can be misleading*". Baxter et al. [4] report that some structural properties of Java software follow power-law distributions, while other do not. They conjecture that metrics measuring properties that programmers are inherently aware about (e.g., out-degree distributions or number of method parameters) tend to follow distributions that are not power-law distributions.

From the observation that source code metrics may follow heavy-tailed distributions, researchers have recently proposed techniques for extracting reliable thresholds for existing metrics, including McCabe [1] and object-oriented metrics [19, 20].

7. CONCLUSION AND FUTURE WORK

We analyzed the statistical distribution of the scattering degrees in five open-source pre-processor-based systems (vi, libxml2, lighttpd, MySQL, and Linux kernel). Our investigation shows that the feature-scattering distribution of four of our subjects (vi, lighttpd, MySQL, and Linux) follow a power law. We provide preliminary evidence that feature scattering is concentrated in specific features that skew the distribution. In such settings, feature-scattering thresholds based on central measures may not be reliable in practice.

To confirm or refute our preliminary findings, researchers shall investigate a larger set of systems. In this direction, we aim at re-applying our methodology in the investigation of other pre-processor-based software families, ideally covering the same systems as Liebig et al. By increasing the target corpus, we also plan to investigate how to extract reliable thresholds for pre-processor-based software families. Another direction for future investigation is to assess feature scattering in software families written in languages other than C (e.g., in object-oriented languages).

8. ACKNOWLEDGEMENTS

We thank CNPq, CAPES, FAPEMIG and German Research Foundation (AP 206/4, AP 206/5, AP 206/6) for partially funding this project.

9. REFERENCES

[1] T. L. Alves, C. Ypma, and J. Visser. Deriving Metric Thresholds from Benchmark Data. In *Proceedings of the International Conference on Software Maintenance*, pages 1–10. IEEE, 2010.

[2] S. Apel, D. Batory, C. Kstner, and G. Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer, 2013.

[3] S. Apel, T. Leich, and G. Saake. Aspectual Feature Modules. *IEEE Transactions on Software Engineering*, 34(2):162–180, 2008.

[4] G. Baxter, M. Frean, J. Noble, M. Rickerby, H. Smith, M. Visser, H. Melton, and E. Tempero. Understanding the Shape of Java Software. In *Proceedings of the International Conference on Object-oriented Programming Systems, Languages, and Applications*, pages 397–412. ACM, 2006.

[5] A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-Law Distributions in Empirical Data. *Society for Industrial and Applied Mathematics Review*, 51(4):661–703, 2009.

[6] G. Concas, M. Marchesi, S. Pinna, and N. Serra. Power-Laws in a Large Object-Oriented Software System. *IEEE Transactions on Software Engineering*, 33(10):687–708, 2007.

[7] M. Couto, M. Valente, and E. Figueiredo. Extracting Software Product Lines: A Case Study Using Conditional Compilation. In *Proceedings of the European Conference on Software Maintenance and Reengineering*, pages 191–200. IEEE, 2011.

[8] M. Eaddy, T. Zimmermann, K. D. Sherwood, V. Garg, G. C. Murphy, N. Nagappan, and A. V. Aho. Do Crosscutting Concerns Cause Defects? *IEEE Transactions on Software Engineering*, 34(4):497–515, 2008.

[9] J.-M. Favre. Preprocessors from an Abstract Point of View. In *Proceedings of the International Conference on Software Maintenance*, pages 329–. IEEE, 1996.

[10] E. Figueiredo, B. C. da Silva, C. Sant'Anna, A. F. Garcia, J. Whittle, and D. J. Nunes. Crosscutting Patterns and Design Stability: An Exploratory Analysis. In *Proceedings of the International Conference on Program Comprehension*, pages 138–147. IEEE, 2009.

[11] C. S. Gillespie. *Fitting Heavy-Tailed Distributions: The poweRlaw Package*, 2014. R package version 0.20.5.

[12] C. S. Gillespie. *The poweRlaw Package: A General Overview*, 2014.

[13] C. Kästner, S. Apel, and K. Ostermann. The Road to Feature Modularity? In *Proceedings of the International Software Product Line Conference*, pages 1–8. ACM, 2011.

[14] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-Oriented Programming*, pages 220–242. Springer, 1997.

[15] G. Krone, M.; Snelting. On the Inference of Configuration Structures from Source Code. In *Proceedings of the International Conference on Software Engineering*, pages 49–57. IEEE, 1994.

[16] J. Liebig, S. Apel, C. Lengauer, C. Kästner, and M. Schulze. An Analysis of the Variability in Forty Preprocessor-Based Software Product Lines. In *Proceedings of the International Conference on Software Engineering*, pages 105–114. ACM, 2010.

[17] J. Liebig, C. Kästner, and S. Apel. Analyzing the Discipline of Preprocessor Annotations in 30 Million Lines of C Code. In *Proceedings of the International Conference on Aspect-Oriented Software Development*, pages 191–202. ACM, 2011.

[18] P. Louridas, D. Spinellis, and V. Vlachos. Power Laws in Software. *ACM Transactions on Software Engineering and Methodology*, 18:1–26, 2008.

[19] P. Oliveira, F. Lima, M. T. Valente, and S. Alexander. RTTOOL: A Tool for Extracting Relative Thresholds for Source Code Metrics. In *Proceedings of the International Conference on Software Maintenance and Evolution (Tool Demo Track)*, pages 1–4, 2014.

[20] P. Oliveira, M. Valente, and F. Paim Lima. Extracting Relative Thresholds for Source Code Metrics. In *Proceedings of the International Conference on Software Maintenance, Reengineering and Reverse Engineering*, pages 254–263. IEEE, 2014.

[21] L. Passos, J. Guo, L. Teixeira, K. Czarnecki, A. Wąsowski, and P. Borba. Coevolution of Variability Models and Related Artifacts: A Case Study from the Linux Kernel. In *Proceedings of the International Software Product Line Conference*, pages 91–100. ACM, 2013.

[22] H. Spencer and G. Collyer. #ifdef Considered Harmful, or Portability Experience with C News. In *Proceedings of the USENIX Technical Conference*, page 185–197. USENIX Association, 1992.

[23] K. Sullivan, W. G. Griswold, Y. Song, Y. Cai, M. Shonle, N. Tewari, and H. Rajan. Information Hiding Interfaces for Aspect-Oriented Design. In *Proceedings of the International Symposium on Foundations of Software Engineering*, pages 166–175. ACM, 2005.

[24] R. Wheeldon and S. Counsell. Power Law Distributions in Class Relationships. In *Proceedings of the International Working Conference on Source Code Analysis and Manipulation*, pages 45–54. IEEE, 2003.