Software Evolution and Time Series Volatility: An Empirical Exploration

Jukka Ruohonen
Department of Information Technology, University of Turku, FI-20014 Turun yliopisto, Finland
juanruo@utu.fi

Sami Hyrynsalmi
Department of Information Technology, University of Turku, FI-20014 Turun yliopisto, Finland
sthyry@utu.fi

Ville Leppänen
Department of Information Technology, University of Turku, FI-20014 Turun yliopisto, Finland
ville.leppanen@utu.fi

ABSTRACT

The paper presents the first empirical study to examine econometric time series volatility modeling in the software evolution context. The econometric volatility concept is related to the conditional variance of a time series rather than the conditional mean targeted in conventional regression analysis. The software evolution context is motivated by relating these variance characteristics to the proximity of operating system releases, the theoretical hypothesis being that volatile characteristics increase nearby new milestone releases. The empirical experiment is done with a case study of FreeBSD. The analysis is carried out with 12 time series related to bug tracking, development activity, and communication. A historical period from 1995 to 2011 is covered under a daily sampling frequency. According to the results the time series dataset contains visible volatility characteristics, but these cannot be explained by the time windows around the six observed major FreeBSD releases. The paper consequently contributes to the software evolution research field with new methodological ideas, as well as with both positive and negative empirical results.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*Process metrics*

General Terms

Economics, Experimentation, Measurement

Keywords

Software evolution, code churn, time series analysis, volatility, conditional variance, ARIMA, GARCH, FreeBSD

1. INTRODUCTION

This exploratory empirical paper has a dual purpose (P): to introduce and evaluate time series volatility modeling for software evolution research (P_1), and to investigate whether volatility increases near new operating system releases (P_2).

Volatility is part of the econometric time series nomenclature. While no unequivocal definitions can be given for the phenomenon, some aspects of volatility are generally agreed upon particularly in financial econometrics; volatility is related to risks, uncertainty, and variance. Accordingly, variability depends on past variability, which causes uncertainty and risks in planning and estimation. In stock markets a conventional reasoning starts from the serially correlated arrival of new and unanticipated news available to the traders [1, 2]. The actors and their algorithms will have different reactions to these news; someone or something may sell, someone may buy. This causes rapid short-run fluctuations. As someone or something may subsequently buy or sell as a response to the increased activity, volatility tends to cluster in time. Once the buyers and sellers have amalgamated the new news and each other's reactions, trading returns to its normal trajectory. The message from this naïve sketch for an economic interpretation is that the interest in volatility modeling is to observe activity, volatility, rather than the time series averages, or the direction of time series.

Volatility is likely to be observable in numerous different software evolution applications [3], although the generative theoretical mechanisms differ. It can be related to the modeling of email flows, for instance. If *volume* measures the number of emails, *velocity* would measure how rapidly the volume of emails changes [4]. This is also one theoretical rationale for volatility modeling: early resolution of uncertainty helps to plan the future, but the point in volatility modeling is not necessarily whether the release of information will result changes, but rather whether the acceleration in the flow of information will result changes [5]. If change rates, or first-differences, $\Delta y_t = y_t - y_{t-1}$, are observed, increasing volatility would then indicate increasing velocity through increasing variance in the change rates. The econometric volatility effect is also stochastic; a large change in volume is likely to be followed by another large change.

Perhaps the most apparent long-run software evolution explanation relates to new milestone releases around which software engineering and related activities tend to increase or intensify [6, 7]. Schedules are set, final development iterations are completed, increasing variability is introduced by branching [8], reliability assessments are made, marketing scenarios are planned or fine-tuned, and so forth. Since code *churn* presumably increases temporally as a result, the probability of making mistakes likely increases, including the like-

IWPSE'15, August 30, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3816-5/15/08...$15.00
http://dx.doi.org/10.1145/2804360.2804367

lihood to introduce security vulnerabilities [9, 10, 11]. Besides these software engineering aspects, volatility may be important for marketing purposes. In the age of crowdsourcing and fundraising, knowing when a crowd reacts may be important in targeting the timing of marketing and other visibility campaigns, for instance.

The traditional software evolution background would also relate to the continuous changes, which should not be too rapid in order to maintain developers' ability to absorb the new information related to a code base [12]. As will be later elaborated in detail, volatility tends to cluster in time. In a sense, therefore, volatility works against the principle of constant and stable change rates. The effect may be seen in different short-run sliding windows that measure larger commit transactions in version control systems [13]. Software engineering processes offer another example. As volatility implies time-dependencies in the variance of a time series, common practices such as sprints in agile development are likely to cause different volatility characteristics; variance may start to cluster towards the end of a sprint, for instance.

In a summary, there are numerous different software engineering aspects that may lead to time series volatility. While the paper leaves the theoretical interpretation open, the empirical rationale is clear: an empirical baseline is required for volatility modeling before explicit, theoretically meaningful software engineering questions can be investigated in more detail (P_1). Nevertheless, the timing of major operating system releases is used to theoretically motivate the empirical experiment that models volatility in twelve high-frequency time series extracted from the FreeBSD development infrastructure (P_2). The expectation is that volatility increases in the proximity of the six major FreeBSD releases made between 1995 and 2011. Before proceeding to the actual experiment, the empirical approach is first elaborated.

2. EMPIRICAL APPROACH

The empirical approach can be elaborated by first considering the basic time series rationale behind econometric volatility modeling. The proximity of releases is subsequently discussed in relation to the models. The fitting strategy follows.

2.1 Volatility

The time series volatility phenomenon is best illustrated by considering simple algebraic assumptions in time series regression models. Therefore, consider a model:

$$y_t = f(y_t \mid \mathcal{I}_{t-}) + \epsilon_t = \mathbf{x}_t'\boldsymbol{\beta} + \epsilon_t, \quad t = 1, \ldots, T, \quad (1)$$

where y_t is the dependent (modeled) time series, $\mathbf{x}_t'$ is a vector of independent variables, $\boldsymbol{\beta}$ is a vector of regression coefficients, and ϵ_t is the residual process. The function $f(\cdot)$ is used to denote an arbitrary time series model that is based on the information set, $\mathcal{I}_{t-}$, available to model estimators before the current time index t. The vector $\mathbf{x}_t'$ may, therefore, contain also lagged values of y_t, among other explanatory variables.

For instance, and for the purposes of this paper, the function $f(\cdot)$ can be assumed to denote the classical autoregressive (AR) integrated (I) moving-average (MA) model (ARIMA). Given non-negative integers p, d, and q for the order of the abbreviations AR, I, and MA, respectively, the general goal is to have a small parsimonious ARIMA(p, d, q) model that renders ϵ_t as white noise while providing good predictive power. While further acronyms and different alterations are often required, this classical model generally performs even amazingly well in forecasting [6, 14]. The empirical rather than theoretical motivation behind the ARIMA model is illustrative also from a different viewpoint.

The goal in ARIMA-like modeling is to forecast the future. In the context of the regression equation (1) this implies that the model predicts the *conditional mean*,

$$\mathrm{E}(y_t \mid \mathbf{x}_t') = \mathbf{x}_t'\boldsymbol{\beta}, \quad \mathbf{x}_t' \in \mathcal{I}_{t-}. \quad (2)$$

If the model fits well for a given dataset, assumptions

$$\mathrm{E}(\epsilon_t) = 0, \quad \mathrm{Var}(\epsilon_t) = 1, \quad \text{and} \quad \epsilon_t \sim \mathrm{N}(0, 1) \quad (3)$$

are often made. In other words, ϵ_t is independent and identically distributed from the normal distribution with a mean of zero and variance equal to unity; the residual process is Gaussian white noise. Time series volatility models are interested in the cases for which these assumptions do not apply. More specifically, the interest is to model the conditional variance rather than the conditional mean.[1]

Consider now that the model in (1) instead yields a residual process υ_t that contains the white noise ϵ_t from (3), and another component that is dependent on the past:

$$\upsilon_t = \epsilon_t\sigma_t, \quad (4)$$

where σ_t denotes the positive time-varying component of interest [15]. This provides the underlying idea in volatility modeling: the goal is to condition this component based on the information available in $\mathcal{I}_{t-}$. In terms of (4) this means that the white noise process is multiplied by the conditional standard deviation, σ_t, which is always non-negative.[2]

Thus, from a Bayesian viewpoint, the idea is to make dependent draws from the normal distribution rather than independent draws from $\sim \mathrm{N}(0, 1)$, that is, from the standard normal distribution. Although normality as such is not important, this amounts to saying that

$$\upsilon_t \mid \mathcal{I}_{t-} \sim \mathrm{N}(0, \sigma_t^2) \quad (5)$$

or that

$$y_t \mid \mathcal{I}_{t-} \sim \mathrm{N}[\mathbf{x}_t'\boldsymbol{\beta}, \mathrm{Var}(y_t \mid \mathcal{I}_{t-})], \quad (6)$$

where $\mathbf{x}_t'\boldsymbol{\beta}$ is the conditional mean from (2). In other words, both the conditional mean and the conditional variance are parameterizable functions of the information in $\mathcal{I}_{t-}$.

The statistical meaning of volatility can be now contemplated by comparing the assumption $\sim \mathrm{N}(0, 1)$ in (3) to the volatility assumption in (5). Since the residual process from (1) now has a variance σ_t^2 that can be predicted by the available information before t, the conditional variance evolves in time. For instance, the variance might be dependent on the past variances at $t-1$ and $t-2$,

$$\mathrm{Var}(\upsilon_t \mid \upsilon_{t-1}, \upsilon_{t-2}) = \sigma_t^2, \quad (7)$$

or more generally

$$\sigma_t^2 = \mathrm{Var}(\upsilon_t \mid \mathcal{I}_{t-}) = \mathrm{Var}(y_t \mid \mathcal{I}_{t-}). \quad (8)$$

[1] In terms of the existing classification of software evolution volatility into amplitude, periodicity, and dispersion [3], econometric volatility modeling would concentrate to the last type. Note, though, that for instance periodicity can be incorporated to the econometric volatility models.

[2] Notice that the unconditional mean of υ_t still remains constant; taking expectations from (4) yields zero due to the white noise assumptions in (3).

An intuitive interpretation would be that in case the past drews, $\sim \mathrm{N}(0, \sigma_{t-1}^2)$ and $\sim \mathrm{N}(0, \sigma_{t-2}^2)$, were done by specifying a large dispersion for the normal probability distribution, then also σ_t^2 is likely to be large. This explains why volatility tends to cluster in time. This tendency is neither deterministic nor infinite, however. Although the dependence on time typically fades away quickly, a large variance does not translate always to a large variate [16]. In other words, even if σ_{t-2}^2 and σ_{t-1}^2 were large, the variance at t may sometimes be moderate. Nevertheless, a time series *shock* – such as a new release – will show as a large deviation of the dependent process from the specified conditional mean (i.e., as a deviation of y_t from the fitted values), but the same shock can be seen also as a large positive or negative value in the residual process [17]. Volatility models do not make assumptions about the signs of the residuals, however. In other words, large values in the residual process tend to be followed with large values of either sign.

This statistical meaning also underlines the theoretical weaknesses in volatility modeling; what does it mean to model the conditional variance? In financial econometrics volatility modeling became extremely popular supposedly largely because of direct theoretical applicability and value in practical policy decisions. In other fields the formulation of theory around the processes has often been more difficult. One way to approach the question is technical. It has been suspected, among other things, that the processes such as (7) might be simply due to the effect of variables omitted from estimated models [15]. This portrays the processes in usual statistical terms; as evidence of misspecification and nuisance to get rid of. In software evolution this rationale would perhaps lead to smoothing with different time series filters [6, 18], given the general empirical software engineering rationale to clean datasets from undesirable noise [19], and to avoid the omitted variable bias [20]. If the volatility noise is retained, however, it is often unclear in applied work whether the theoretical shocks should be modeled in terms of the conditional mean or the conditional variance. The empirical experiment follows the latter path; the new major releases are assumed to shock the second conditional term in (6). If this theoretical hypothesis is rejected, the potential volatility characteristics (P_1) are likely generated by other mechanisms than release engineering and associated activities (P_2). If the hypothesis holds, the concept of *time deformation* [1, 21] could be adopted as a high-level theoretical explanation; the real calendar time and the release-cycle pseudo-time may proceed at different speeds, generating different volatility characteristics.

2.2 The Proximity of Releases

Consider the following simple definition for a deterministic control variable PRX_t^r, the proximity of the r:th release:

$$\mathrm{PRX}_t^r(t_a, t_r, t_b) = \begin{cases} 1 & \text{if} \quad t_a \leq t_r \leq t_b \\ 0 & \text{if} \quad t_r \notin [t_a, t_b] \end{cases}, \tag{9}$$

where all three date indices are inside a time series sample window, and t_r denotes an index of a given release day. A case $t_a = t_r = t_b$ defines a standard dummy variable, omitting the word proximity present in the abbreviation. That is, only the actual release day would be observed.

The software engineering motivation for (9) comes from the sliding window model that was initially introduced for the detection of commit transaction sequences and associated bursts [13], and then extended to other purposes with good results [22, 23]. The general idea is also rather similar to the windows used in the so-called event studies that examine stock market reactions to different unanticipated events, including, for instance, mergers and acquisitions, computer security breaches, and outsourcing events and announcements [2, 24]. These have been incorporated also to volatility modeling [1]. Regardless of the domain of application, there is a generic problem associated with all simple, deterministic windows: the length cannot be easily defined. While different transition state models [25] could be considered, the problem can be approached also theoretically.

Although large variations are present between different open source projects, existing empirical research indicates that traffic increases near adoption peaks (releases), while traffic in development-oriented systems increases before releases or large refactoring periods [26]. The latter effect is typical particularly in the BSD operating system context within which releases are polished for a relatively long period by the developer community, although a release engineering team typically handles the actual scheduling and branching. The preparation periods include, for instance, different "hackathons", which hint that commits and general programming activity increase before releases. However, provided that adoption peaks nearby major releases, also systems such as bug tracking should show increasing volatility as bugs are discovered by the newly arrived users. Further examples are easy to pinpoint to signify this heterogeneity assumption. For instance, the arrival of social media has partially challenged the existing mailing list communication media; open source blogging has been observed to increase after releases, partially again because of the non-developer communities [7]. All in all, it seems that volatility is likely to vary from a software development work system to another, but it is more difficult to make clear-cut theoretical corollaries regarding the timings.

To make the theoretical assumptions explicit: the window in (9) is defined with a positive integer that is uniformly shared between the observed $r = 1, \ldots, R$ releases. That is, a *modified proximity metric* $\tilde{\mathrm{PRX}}_t^r$ takes a value one in case $t_r \in [t_r - k, t_r + k]$ for some shared positive integer $k > 0$, $t_r + k < T, t_r - k > 1$. For instance, with $k = 1$ a metric $\tilde{\mathrm{PRX}}_t^r(k, t_r, k)$ yields a vector $(0, \ldots, 0, 1, 1, 1, 0, \ldots, 0)$, which would be presumed to capture the shock introduced by the r:th release in the conditional variance, given a model for the conditional mean.

2.3 Estimation

The seminal work in [15] introduced an autoregressive conditional heteroskedasticity model, denoted by ARCH(q). The key idea behind the ARCH(q) model is that υ_t in (4) is uncorrelated across time with $\mathrm{E}(\upsilon_t) = 0$, but the conditional variance of it, σ_t^2, may change in time. The original model required weights to ensure that volatility will eventually decay. This motivated the generalized ARCH representation known as GARCH. This GARCH(p, q) model, which was introduced in [27], is based on a seemingly ARMA-like idea:

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \upsilon_{t-i}^2 + \sum_{j=1}^{p} \beta_j \sigma_{t-j}^2, \tag{10}$$

where $q > 0$, $p \geq 0$, $\alpha_0 > 0$, and $\alpha_i \geq 0$ for all $i = 1, \ldots, q$, and $\beta_j \geq 0$ for all $j = 1, \ldots, p$. These restrictions en-

sure that the conditional variance is always strictly positive. An important mathematical result relates to the unconditional variance that should be asymptotically constant in order for the process to be stationary [15, 27]. For instance, given the non-negativity restriction required for variance, a GARCH(1, 1) process is (weakly) stationary if $\alpha_1 + \beta_1 < 1$.

The model building process is similar to ARIMA modeling. In fact, the volatility models are typically fitted by using an ARMA structure for the mean and a GARCH structure for the variance, resulting GARCH(p_1, q_1)-ARIMA(p_2, d, q_2) models. The identification is often easier for the variance parameters p_1 and q_1 than for the conventional AR(p_2) and MA(q_2) terms. For instance, it has been observed that a short GARCH(1, 1) structure is adequate to represent many financial time series [17]. This was used as the reference point in the empirical experiment. The ARMA structures were constructed by visually investigating correlograms, and by checking that no remaining autocorrelation was present according to the Ljung-Box test. The (G)ARCH effects can be evaluated with the Engle-test named after the inventor. In essence: given (10) and residuals from (1), a model

$$\widehat{\epsilon}_t^2 = \alpha_0 + \alpha_1 \widehat{\epsilon}_{t-1}^2 + \alpha_2 \widehat{\epsilon}_{t-2}^2 + \cdots + \alpha_q \widehat{\epsilon}_{t-q}^2 + u_t, \qquad (11)$$

should fit well in case there are volatility characteristics in y_t. Alternatively, the Ljung-Box test can be used to investigate the effect of the past squared residuals.

There are at least two approaches to model the release windows. The first is based on the conventional machine learning principle: given a predefined period before the t_r:th release, train the data in a window before t_r, and then carry out the evaluation in the window starting from $t_r + k$.[3] This has been also a common principle in the noted event studies [28], although in time series analysis numerically equivalent results can be (often) obtained by using deterministic dummy variables [29, 30]. The latter path is suitable for the purposes of this paper. Consequently, the modified $\tilde{\mathrm{PRX}}_t^r$ metric was first used to investigate the effect of each of the six major releases separately. The results were similar to using the following additional simplification:

$$\widetilde{\mathrm{PRX}}(k) = \sum_{j=1}^{6} \tilde{\mathrm{PRX}}^{r_j}(k, t_{r_j}, k), \qquad (12)$$

which aggregates the individual effects into a single vector. This is used in the results reported. The following windows were tested: $k = 1$ (one day before and after), $k = 3$ (a week), and $k = 15$ (approximately a month). The results were similar with all integers. Finally, the *rugarch* package [31] is used for computation.

3. EMPIRICAL EXPERIMENT

The empirical experiment proceeds by first introducing the FreeBSD dataset. A few statistical observations are subsequently made. The volatility estimates follow.

3.1 Data

[3] Although the estimation can be done for instance by considering the estimation window, the event window, and the post-event window [28], it should be emphasized that the time series context forbids randomization of observations in the windows. That is to say, the approach should not be confused with the k-fold cross-validation techniques.

The twelve time series metrics are summarized in Table 1. All series are observed daily during a period from the first day of January 1995 to the last day of December 2011. Following the existing practices [32], all metrics were collected on *a priori* grounds, meaning that the selected features were chosen before analysis. This choice also limited the end of the sampling period to 2011. The time series metrics are grouped into four categories: bug tracking, mailing lists, commit activity, and commit magnitudes.

Two major changes have been made in the FreeBSD project regarding the essential software development tools. First, the Concurrent Versions System (CVS) was replaced with Subversion in 2008. All data was extracted from the transformed Subversion histories. Second, the project decided to abandon the old email-based GNATS bug tracking system in 2014. The bug tracking data refers only to the historical tracking with the old system. The two metrics BRA and BRC, the number of arrived and closed bug reports, respectively, are based directly on data extracted from GNATS. No classifications are mare according to the status, type, or category of bugs. The third bug tracking metric, BRR, records the number of replies made to the reports. Given the way GNATS works [33], BRR was matched from replies to the `freebsd-bugs` mailing list by using a regular expression.

Mailing lists provide the primary medium for epistemic interactions [34] particularly in the open source operating system context. Four mailing list archives were consequently included in the sample: besides the noted `freebsd-bugs`, `freebsd-current` (discussions concerning the running development branch), `freebsd-hackers` (general technical discussions and debates), and `freebsd-questions` (user questions and support) are observed. These refer to BRR, MLC, MLH, and MLQ in Table 1, respectively.

Development activity is observed with six metrics grouped into two categories, both of which relate to code churn [11]. First, the three commit activity metrics simply count the number of commits. The three activity metrics (CTB, CTP, and CTD) differ in their domain, however. FreeBSD contains three separate development domains with three separate version control trees: `base` refers to the actual BSD operating system; `ports` is used for packing external third-party open source software; and `doc` is used for documentation. Second, the analogously grouped metrics LDB, LDP, LDD were obtained commit-by-commit with `svnlook diff`, which was passed to `wc -l`. Consequently, the metrics measure *roughly* the total size or magnitude of commits in terms of the changed files and lines in a day. These are, of course, only approximations; there is a large and systematic divergence from the magnitude of changed source code already due to the additional information printed by Subversion. Since the empirical interest (P_2) is to model volatility – for which churn seems like the ideal case – the interpretations should not be significantly threatened by these inaccuracies.

All collected metrics were aggregated to cumulative daily sums. In the time series literature this aggregation solution is sometimes referred to as a *flow* scheme, as opposed to a *stock* or aggregation by using different averages [35]. More importantly, the time series are not irregular but instead follow equally spaced, daily intervals in calendar time. Given four leap years, each time series $i = 1, \ldots, 12$ has a length of $T = (4 \times 366) + (13 \times 365) = 6209$. No data transformations were applied, although the three commit magnitude metrics

Table 1: Metrics

1. Bug Reports		3. Commit Activity	
BRA	Number of arrived bug reports	CTB	Number of commits to `base`
BRC	Number of closed bug reports	CTP	Number of commits to `ports`
BRR	Number of replies to bug reports	CTD	Number of commits to `doc`
2. Mailing Lists		**4. Commit Magnitudes**	
MLC	Number of messages on `current`	LDB	Lines of commit differences in `base`
MLH	Number of messages on `hackers`	LDP	Lines of commit differences in `ports`
MLQ	Number of messages on `questions`	LDD	Lines of commit differences in `doc`

The bug tracking and mailing list data is based on archives that were provided by the FreeBSD project in July 2014 from the server at `ftp://ftp.freebsd.org`. The commit data is based on full Subversion trees provided from the same server for the purposes of mirroring.

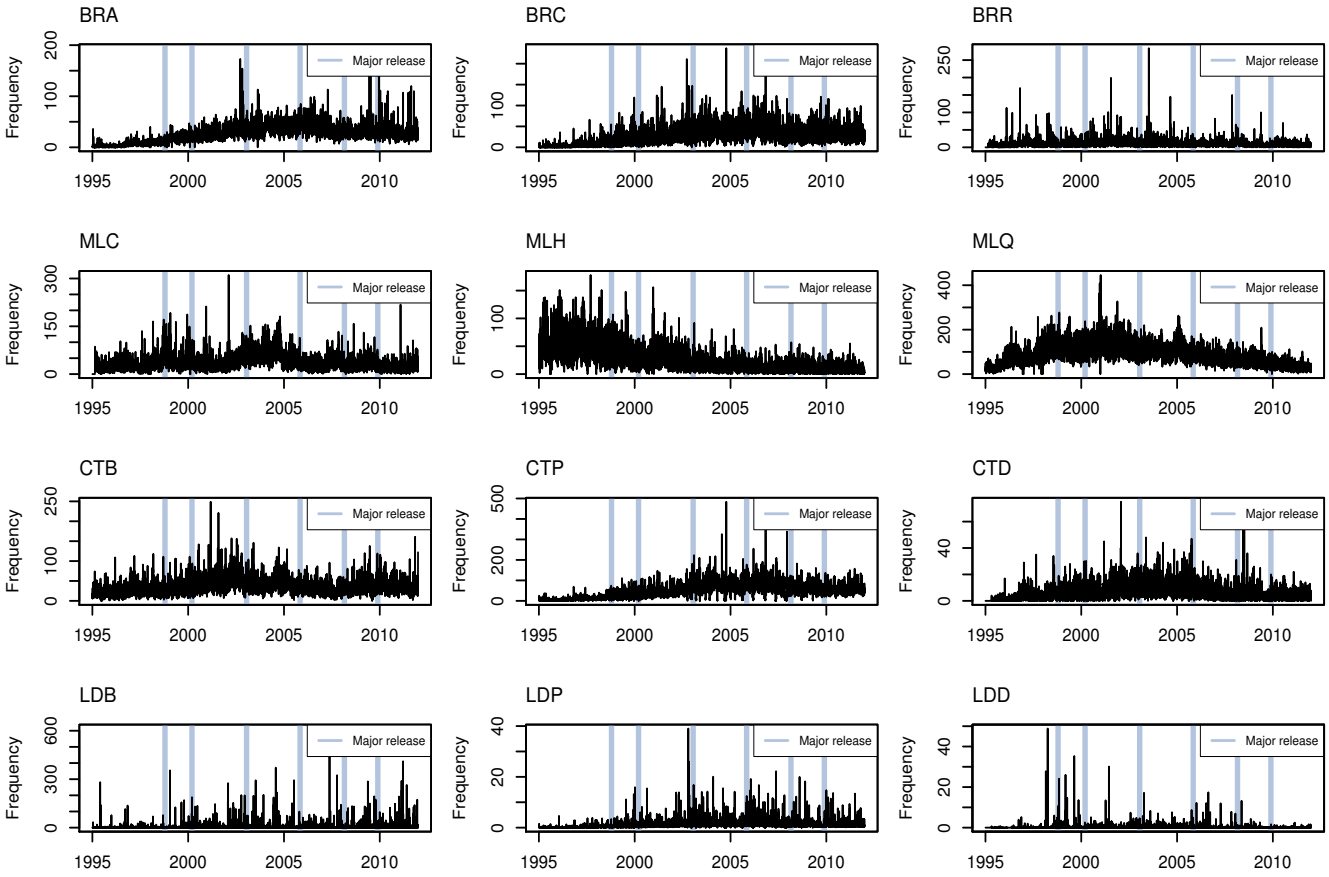


Figure 1: Data

were multiplied by 10000^{-1} for easier interpretation. This has no effect on the results reported.

3.2 Preliminaries

The metrics are visualized in Fig. 1. In general, the laws of software evolution [12] have been observed to hold also in FreeBSD. The continuous growth of the code base has been observed to follow a linear trend, for instance [36]. The six commit activity and magnitude metrics generally support the observation; the development activity has not halted. Most of the time series show significant volatility noise, however, which prevents a human eye from clearly seeing the time series trends. If the kernel smoothing techniques [6] are used to aid the visual inspection, several different trends become more visible. For instance, the trend in MLQ shows an interesting boomerang-shape. Another point worth remarking relates to the trends in the bug tracking system (see Fig. 2). The arrival of new bug reports seems to have closely followed the closure of old reports in the long-run.

On the other hand, even after filtering a human eye has difficulties in seeing a clear clustering pattern around the six major releases. There are some peaks, however. For instance, some series peak near the fifth major release. This historical release was developed in the first part of the 2000s

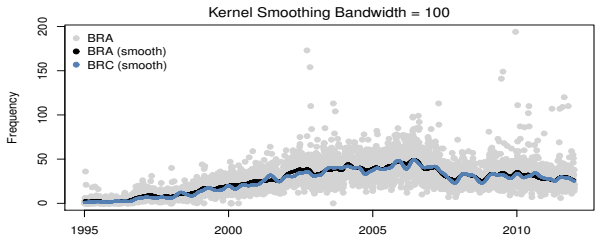


Figure 2: Trends in BRA and BRC

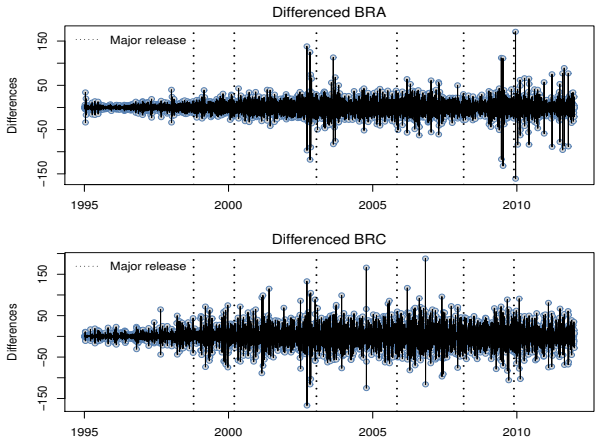


Figure 3: Differenced BRA and BRC

during which symmetric multiprocessing was introduced to FreeBSD, among other large architectural changes that also caused some community problems [37]. In general, however, the visible turbulence periods do not seem to systematically cluster around the proximity of all observed releases.

Other basic time series characteristics can be summarized with three general observations. First, volatility models assume *stationarity*. Formal test results are consequently reported in Table 2. The null hypothesis of stationarity is clearly rejected for all series, while all differenced series appear to be stationary. A further concern relates to the visible level shifts in the variance trajectories (see Fig. 1), which are typical factors that may undermine the stationary assumption [16], leading to the so-called integrated I-GARCH model [38]. Nevertheless, the analysis is carried out under the assumption that the first-differenced series are adequate for the fitted GARCH-AR(1)MA models. To illustrate the graphical shape of the first-differenced series, the differenced BRA and BRC are shown in Fig. 3.

Second, *normality* is a potential problem. As might be typical for software engineering time series, all differenced series are distributed according to a bell-shaped curve that somewhat resembles the normal distribution, although the mass is sharply located around zero. Formal tests, consequently, reject normality for all univariate series, whether in levels or in differences (see Table 3). This suggests that some alternative distribution could be preferable. In volatility modeling the Student's t-distribution has been a popular alternative for the normal distribution [16, 31]. It also seems like a reasonable choice to describe the density shapes of the differenced series. To investigate this possibility further, Fig. 4 shows quantile-quantile plots for the differenced BRA and BRC using the (central) Student's t-distribution. A visual conclusion confirms that the normality assumption may not be realistic. Consequently, the t-distribution is specified for the reported models, although the same conclusions can be reached under the more familiar normal distribution.

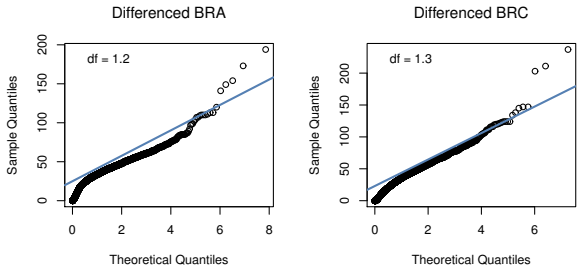


Figure 4: Two Student's t Q-Q Plots

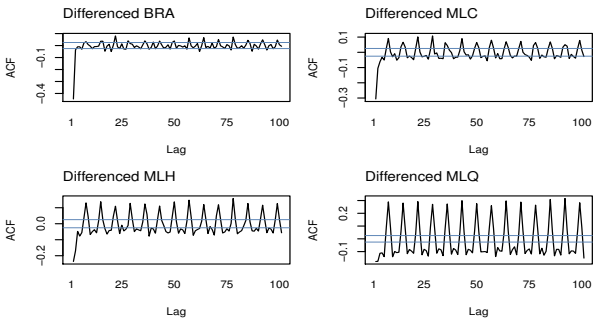


Figure 5: Four Correlograms

Third, the three mailing list time series exhibit *seasonality*. This can be illustrated with the four correlograms in Fig. 5. (All other series show similar patterns to the shown differenced BRA). The cycle seems to follow relatively clearly the weekly variation. While seasonal ARIMA models [14] could be considered for the conditional mean models, no special seasonal effects are included for the three mailing list metrics.[4] The autocorrelation functions (ACFs) also indicate that the memory of the differenced series seem to be relatively short in general. Moreover, differencing seems to cause a negative autocorrelation at the first lag, suggesting that simple first-differences do not fully account for the trending characteristics in the level of the series.

3.3 Results

The formal modeling can be started by investigating the time-dependence of the residual variance processes. This can be done in two simple steps. First, plain AR(p) models were fitted for the first-differenced series by letting the Akaike's information criterion (AIC) to pick the suitable lag length automatically. Second, the squared residuals from the autoregressive results were used to estimate (11) by again

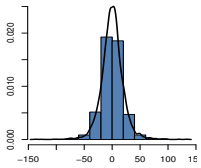
[4] The reason is partially practical; the used implementation [31] does not allow to easily define seasonal components.

Table 2: Unit Root Tests (KPSS)

	Levels		Differences			Levels		Differences	
	Value	Result	Value	Result		Value	Result	Value	Result
BRA_t	11.42	$\sim I(1)$	0.01	$\sim I(0)$	CTB_t	2.25	$\sim I(1)$	0.02	$\sim I(0)$
BRC_t	11.18	$\sim I(1)$	0.01	$\sim I(0)$	CTP_t	11.76	$\sim I(1)$	< 0.01	$\sim I(0)$
BRR_t	1.20	$\sim I(1)$	0.01	$\sim I(0)$	CTD_t	3.84	$\sim I(1)$	0.01	$\sim I(0)$
MLC_t	1.52	$\sim I(1)$	0.01	$\sim I(0)$	LDB_t	3.33	$\sim I(1)$	< 0.01	$\sim I(0)$
MLH_t	14.46	$\sim I(1)$	0.01	$\sim I(0)$	LDP_t	7.72	$\sim I(1)$	< 0.01	$\sim I(0)$
MLQ_t	5.59	$\sim I(1)$	0.02	$\sim I(0)$	LDD_t	1.18	$\sim I(1)$	< 0.01	$\sim I(0)$

The table reports results from the so-called KPSS stationarity tests [39]. The null hypothesis is a stationary process, denoted by $\sim I(0)$. The alternative of non-stationarity is marked with $\sim I(1)$. More specifically: a tested series x_t is assumed to follow $x_t = z_t + u_t$, where $z_t = z_{t-1} + \epsilon_t$ and $\epsilon_t \sim \mathrm{N}(0, \sigma_\epsilon^2)$. Given this representation, H_0 is that $\hat{\sigma}_\epsilon^2 = 0$, implying that the additional pure random walk term, z_t, is constant around zero mean. The ancillary lag length was truncated from $4 \times \sqrt[4]{(T/100)}$ to 33. Critical values at ten, five, and one percent levels are 0.35, 0.46, and 0.74, respectively, as given by the `ur.kpss` function used for estimation [40].

Table 3: Normality Tests (Shapiro-Wilk)

	Levels		Differences			Levels		Differences		
	Value	p	Value	p		Value	p	Value	p	Density of ΔCTB_t
BRA_t	0.95	✓	0.87	✓	CTB_t	0.91	✓	0.96	✓	
BRC_t	0.90	✓	0.92	✓	CTP_t	0.92	✓	0.86	✓	
BRR_t	0.61	✓	0.75	✓	CTD_t	0.83	✓	0.90	✓	
MLC_t	0.88	✓	0.93	✓	LDB_t	0.24	✓	0.38	✓	
MLH_t	0.84	✓	0.95	✓	LDP_t	0.53	✓	0.60	✓	
MLQ_t	0.95	✓	0.98	✓	LDD_t	0.14	✓	0.24	✓	

The table shows results from the Shapiro-Wilk test; the symbol ✓ marks cases in which H_0 of normality is rejected at 0.001 level. Due to limitations in R's `shapiro.test`, each result is an average from a hundred random samples of size 5000 (from $T = 6209$). The densities of all differenced metrics are roughly similar to the shown density of the differenced amount of commits made to `base`.

letting AIC to choose the lag length. The results are summarized in Fig. 6. While it is clear that a large p is required to describe the differenced series (the left-hand side plot), many metrics indicate also rather long AR(p) processes for the squared residual terms. Evaluating the estimated AR coefficients reinforces the visual conclusion that there are (G)ARCH effects present in most of the time series.

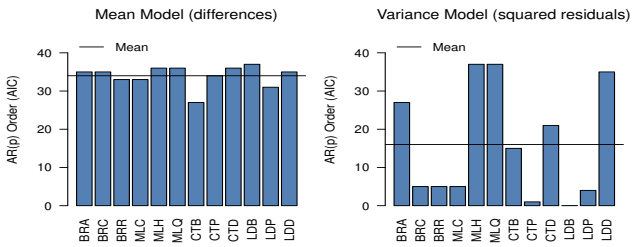


Figure 6: Coarse Volatility Effects

The fitted models are summarized in Table 4. Relatively short GARHC(p_1, q_1) orders are required to control the process in (10) such that no remaining effects are present. Only a few series required higher orders than $p_1 = q_1 = 1$ according to the Engle-test. This is a familiar finding from financial econometrics [1, 17]. Also decent ARMA(p_2, q_2) structures can be located relatively effortlessly by trial-and-error. Most of the fitted models are free of remaining autocorrelation. However, the seasonality patterns affect the mailing list estimates for which decent ARMA structures are difficult to formulate to capture the time series characteristics of MLC, MLH, and MLQ. If information criteria measures and parsimonious models are used to rank the estimates, the winner is a model for LDD.

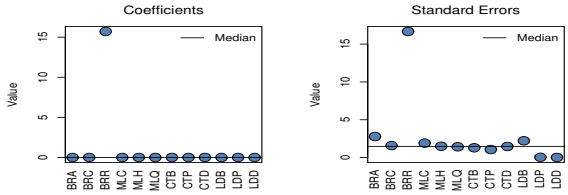


Figure 7: Effect of Release Windows ($k = 3$)

Alas, the proximity of releases has no explanatory power. The coefficients are negligible and the standard errors are large for all but one series (see Fig. 7). The only exception is BRR for which both a very high coefficient and a very high standard error are present. The aggregated window is not statistically significant ($p \simeq 0.35$), however. Widening or shortening the one week windows do not change the results. The same applies to the disaggregated $\mathrm{P\tilde{R}X}_t^r$ windows. It is highly unlikely that the volatility effects can be explained by the proximity of the six major FreeBSD releases (P_2).

Table 4: Volatility Model Diagnostics (GARCH-ARIMA)

	BRA	BRC	BRR	MLC	MLH	MLQ	CTB	CTP	CTD	LDB	LDP	LDD
GARCH(p_1, q_1)	(2, 1)	(1, 1)	(1, 1)	(1, 1)	(2, 1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(0, 1)
ARIMA(p_2, 1, q_2)	(9, 9)	(10, 10)	(2, 2)	(2, 5)	(6, 6)	(10, 10)	(8, 8)	(8, 2)	(8, 2)	(3, 4)	(6, 4)	(1, 1)
Ljung-Box(1)	–	–	–	–	–	–	–	–	–	–	–	–
Ljung-Box(a)	–	–	✓	✓	✓	✓	–	–	–	–	–	–
Ljung-Box(b)	–	–	–	✓	✓	✓	–	–	–	–	–	–
Engle($p_1 + q_1 + 1$)	–	–	–	–	–	–	–	–	–	–	–	–
Engle($p_1 + q_1 + 3$)	–	–	–	–	–	–	–	–	–	–	–	–
Engle($p_1 + q_1 + 5$)	–	–	–	–	–	–	–	–	–	–	–	–
BIC	6.96	7.75	6.69	8.42	7.68	9.09	8.24	8.54	5.38	5.09	2.04	-0.85
No. insignificant	2	2	1	1	2	1	1	2	3	1	2	1
Subjective evaluation	A	A	B	C	C	C	A	A	A	A	A	A

A symbol ✓ denotes $p < 0.05$ in the Ljung-Box test for no autocorrelation (H_0) and in the Engle-test for no remaining ARCH effects (H_0). In both tests the tested lag is shown in parenthesis, given the orders shown in the first two rows. In the former test the lags a and b are defined in [31] as $a = 2 \times (p_2 + q_2) + (p_2 + q_2) - 1$ and $b = 4 \times (p_2 + q_2) + (p_2 + q_2) - 1$. The subsequent row shows the Bayesian information criterion (BIC), followed by the total number of statistically insignificant parameters at 5 % level (using normal, non-robust standard errors). The final row denotes a subjective evaluation according to a threefold criteria: decent (A), moderate (B), and bad (C). All models include the aggregated deterministic variable from (12) with $k = 3$ in the variance (GARCH) model. The models are specified with the Student's t-distribution. The constant term is included in both the mean models and the variance models, although it is insignificant in some models.

The empirical experiment can be completed by demonstrating the predictive potential. The following seems like a theoretically decent example model that might be of interest also to the FreeBSD developers and their bug tracking efforts:

$$\Delta\mathrm{BRC}_t = \mathbf{x}_t'\boldsymbol{\beta} + \epsilon_t, \quad \text{where} \tag{13}$$
$$\Big[\beta_1\Delta\mathrm{BRA}_{t-1}, \beta_2\Delta\mathrm{BRR}_{t-1},$$
$$\beta_3\Delta\mathrm{CTB}_{t-1}, \beta_4\Delta\mathrm{LDB}_{t-1}, \beta_5\widetilde{\mathrm{PRX}}_t(3)\Big] \in \mathbf{x}_t'\boldsymbol{\beta}$$

among past information about $\Delta\mathrm{BRC}_t$. Since no additional information from $\mathcal{I}_{t-}$ is used for the (exogenous) independent variables, the interpretation is relatively clear: the model predicts the current daily changes in closed bug reports by the past changes in arrived bug reports, the past churn in the amount of bug report replies, and the past changes in the development activity within the `base` operating system domain, controlling for the proximity of releases with a weekly window for each release. The direction of causality is not clear, however. For instance, also $\Delta\mathrm{BRA}_t$ could appear on the left-hand side since general development churn is likely to increase the changes in the amount of opened bug reports. Nonetheless, in addition to forecasting, a time series model such as (13) can be used to investigate the factors that affect the closure (or arrival) of bug reports, and to evaluate whether these effects are stable and systematic across time. Since the two metrics BRA_t and BRC_t follow similar long-run trends (see Fig. 2), the general idea for (13) could also be that the two basic bug tracking metrics are intentionally kept synchronized.

The relatively high-order specification for $\Delta\mathrm{BRC}_t$ in Table 4, $p_1 = q_1 = 1$ and $p_2 = q_2 = 10$, can be used to estimate the model. When the additional series are included in the conditional variance equation in (10), the effects are negligible and statistically insignificant. The BIC is 7.72. When the four metrics are instead used in the more conventional conditional mean model, $\widehat{\beta}_1$ and $\widehat{\beta}_3$ are significant at 5 % level with positive signs. Also BIC reduces slightly to 7.69. In other words, some dynamic relationships are present particularly with respect to BRC and BRA, although the effects do not show in the variance trajectories. Regardless whether the additional five metrics are included in the mean model (M_1) or in the variance model (M_2), the sum of the two estimated GARCH coefficients is rather close to unity, suggesting further model calibration to guard for potential violations regarding the stationarity assumption.

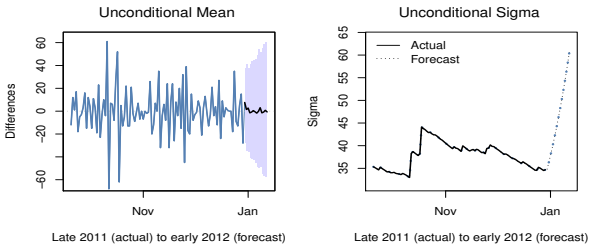


Figure 8: Forecasts (M_1) with (13)

Nevertheless, the underlying forecast setup can be illustrated in the form of Fig. 8, which shows an example of a two week out-of-sample (2012) forecast based on unconditional expectations. As can be seen, the unconditional mean is predicted to remain relatively constant. As the forecast window widens, however, the unconditional expectation for the sigma in (4) starts to increase, indicating increasing uncertainty. This illustrates the practical relevance of volatility modeling: both empirical accuracy and theoretical questions can be approached also by examining the variables and patterns that influence σ_t, the conditional standard deviation.

4. CONCLUSION

It has been widely acknowledged that econometric volatility modeling is only a mechanical way to describe and predict the variance behavior of a time series [15, 16]. It does not provide any explanation on why the variance characteristics tend to rather neatly follow the relatively simple time-dependence pattern elaborated in the paper.

There are some purely empirical explanations, however. One is captured by the econometric saying that volatility cannot be directly observed, which is used to emphasize that aggregation and sampling frequencies hide the more nuanced variability [16]. For instance, this paper used daily sampling frequency, but direct observability would require also assessments over the intra-day variability. Such short-run variability has been well-recognized in the software evolution and engineering research, as demonstrated by the different sliding window and burst models used in the mining of software repositories. This short-run development viewpoint is a typical candidate for further time series volatility modeling. Since the empirical experiment revealed clear daily volatility characteristics also in long-run, decade-wide software evolution time series, it seems reasonable to recommend that formal software evolution modeling should also ascertain that the fitted models are free of volatility effects. This can be done with the elaborated techniques.

The second conclusion relates to the dataset. Since volatility characteristics are present in the FreeBSD bug tracking system as well as mailing lists, it seems reasonable to conclude that churn as a concept is not related only to code. This is the same conclusion that has been previously investigated under a label of interactive socio-technical churn [10]. Not only does code churn increase the probability of mistakes, but the same likely applies to increasing variability in the technical and social software development environment. An interesting modeling context relates to the multivariate volatility models; does increasing volatility in one development system cause time-dependent variability in another system? An analogous further question relates to the surrounding environment; increasing variability in the environment may well translate to software development volatility. Thus, in general, perhaps a more meaningful volatility modeling context relates to the so-called business analytics.

The third conclusion is unambiguous: volatility did not increase in the proximity of the six major FreeBSD releases between 1995 and 2011. Although no direct comparisons can be made, this is a negative result with respect to the existing general empirical interpretations about the intensity near releases [7, 26]. Needless to say, further replicative empirical research would be required to evaluate the reason for this contrary result, or to contemplate whether the result is specific only to the FreeBSD dataset. There is also a twofold confirmation bias present: not only was a predefined set of features selected in advance, but systematic assessments were also omitted regarding the effect of milestone releases upon the conditional mean. The latter point restates the theoretical problems in applied volatility modeling: it is not always clear whether different events or shocks should be modeled in terms of the conditional mean, the conditional variance, or possibly even both.

This negative result does not invalidate the use of volatility modeling in different release engineering scenarios, however. As the paper demonstrates, volatility affects the predictive performance of longitudinal statistical models. If concepts such as time-to-release or time-to-market are considered, it is clear that the timings should not occur during volatility periods as these imply increasing uncertainty. A hectic development period is likely not the time to make judgments about the final software products.

To summarize, volatility modeling is plausible in software evolution research (P_1), although the empirical experiment shows no evidence that volatility would increase in the proximity of new releases (P_2). The likely most challenging further question relates to the theoretical meaning of volatility in software evolution and engineering time series.

5. REFERENCES

[1] T. Bollerslev, R. Y. Chou, and K. F. Kroner, "ARCH Modeling in Finance: A Review of the Theory and Empirical Evidence," *Journal of Econometrics*, vol. 52, no. 1–2, pp. 5–59, 1992.

[2] A. McWilliams and D. Siegel, "Event Studies in Management Research: Theoretical and Empirical Issues," *Academy of Management Journal*, vol. 40, no. 3, pp. 626–657, 1997.

[3] E. J. Barry, C. F. Kemerer, and S. A. Slaughter, "On the Uniformity of Software Evolution Patterns," in *Proceedings of the International Conference on Software Engineering (ICSE 2003)*. Portland: IEEE, 2003, pp. 106–113.

[4] R. S. Debreceny and G. L. Gray, "Data Mining of Electronic Mail and Auditing: Research Agenda," *Journal of Information Systems*, vol. 25, no. 2, pp. 195–226, 2011.

[5] S. A. Ross, "Information and Volatility: The No-Arbitrage Martingale Approach to Timing and Resolution Irrelevancy," *The Journal of Finance*, vol. 44, no. 1, pp. 1–17, 1989.

[6] I. Herraiz, J. M. González-Barahona, G. Robles, and D. M. Germán, "On the Prediction of the Evolution of Libre Software Projects," in *Proceedings of the International Conference on Software Maintenance (ICSM 2007)*. Paris: IEEE, 2007, pp. 405–414.

[7] D. Pagano and W. Maalej, "How Do Open Source Communities Blog?" *Empirical Software Engineering*, vol. 18, no. 6, pp. 1090–1124, 2013.

[8] K. Mohan and B. Ramesh, "Change Management Patterns in Software Product Lines," *Communications of the ACM*, vol. 49, no. 12, pp. 68–72, 2006.

[9] R. M. Bell, T. J. Ostrand, and E. Weyuker, "Does Measuring Code Change Improve Fault Prediction?" in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering (Promise 2011)*. Banff: ACM, 2011, pp. 2:1–2:8.

[10] A. Meneely and O. Williams, "Interactive Churn Metrics: Socio-Technical Variants of Code Churn," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 6, pp. 1–6, 2012.

[11] Y. Shin, A. Meneely, and L. Williams, "Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities," *IEEE Transactions on Software Engineering*, vol. 37, no. 6, pp. 772–787, 2011.

[12] M. M. Lehman, D. E. Perry, and J. F. Ramil, "On Evidence Supporting the FEAST Hypothesis and the Laws of Software Evolution," in *Proceedings of the Fifth International Software Metrics Symposium (METRICS 1998)*. Bethesda: IEEE, 1998, pp. 84–88.

[13] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller, "Mining Version Histories to Guide Software Changes," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 429–445, 2005.

[14] M. Goulão, N. Fonte, M. Wermelinger, and F. B. e Abreu, "Software Evolution Prediction Using Seasonal Time Analysis: A Comparative Study," in *Proceedings of the 16th European Conference on Software Maintenance and Reengineering (CSMR 2012)*. Szeged: IEEE, 2012, pp. 213–222.

[15] R. F. Engle, "Autoregressive Conditional Heteroskedasticity with Estimates of the Variance of United Kingdom Inflation," *Econometrica*, vol. 50, no. 4, pp. 987–1007, 1982.

[16] R. S. Tsay, *Analysis of Financial Time Series*. Chichester: John Wiley & Sons, 2002.

[17] A. K. Bera and M. L. Higgins, "ARCH Models: Properties, Estimation and Testing," *Journal of Economic Surveys*, vol. 7, no. 4, pp. 305–366, 1993.

[18] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus, "Does Code Decay? Assessing the Evidence from Change Management Data," *IEEE Transactions on Software Engineering*, vol. 27, no. 1, pp. 1–12, 2001.

[19] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, A. Ihara, and K. Matsumoto, "The Impact of Mislabelling on the Performance and Interpretation of Defect Prediction Models," in *Proceedings of the 37th IEEE International Conference on Software Engineering (ICSE 2015)*. Florence: IEEE, 2015.

[20] M. Jørgensen and B. Kitchenham, "Interpretation Problems Related to the Use of Regression Models to Decide on Economy of Scale in Software Development," *Journal of Systems and Software*, vol. 85, no. 11, pp. 2494–2503, 2012.

[21] J. H. Stock, "Measuring Business Cycle Time," *Journal of Political Economy*, vol. 95, no. 6, pp. 1240–1261, 1987.

[22] N. Bettenburg and A. E. Hassan, "Studying the Impact of Social Interactions on Software Quality," *Empirical Software Engineering*, vol. 18, no. 2, pp. 375–431, 2013.

[23] C. Lokan and E. Mendes, "Investigated the Use of Duration-Based Moving Windows to Improve Software Effort Prediction: A Replicated Study," *Information and Software Technology*, vol. 56, no. 9, pp. 1063–1075, 2014.

[24] Y. Konchitchki and D. E. O'Leary, "Event Study Methodologies in Information Systems Research," *International Journal of Accounting Information Systems*, vol. 12, no. 2, pp. 99–115, 2011.

[25] J. Kleinberg, "Bursty and Hierarchical Structure in Streams," *Data Mining and Knowledge Discovery*, vol. 7, no. 4, pp. 373–397, 2003.

[26] S. Gala-Pérez, G. Robles, J. M. González-Barahona, and I. Herraiz, "Intensive Metrics for the Study of the Evolution of Open Source Projects: Case Studies from Apache Software Foundation Projects," in *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR 2013)*. San Francisco: IEEE, 2013, pp. 159–168.

[27] T. Bollerslev, "Generalized Autoregressive Conditional Heteroskedasticity," *Journal of Econometrics*, vol. 31, no. 3, pp. 307–327, 1986.

[28] A. C. MacKinlay, "Event Studies in Economics and Finance," *Journal of Economic Literature*, vol. 35, no. 1, pp. 13–39, 1997.

[29] S. Acharya, "Value of Latent Information: Alternative Event Study Methods," *The Journal of Finance*, vol. 48, no. 1, pp. 363–385, 1993.

[30] G. V. Henderson, Jr., "Problems and Solutions in Conducting Event Studies," *The Journal of Risk and Insurance*, vol. 57, no. 2, pp. 282–306, 1990.

[31] A. Ghalanos, "rugarch: Univariate GARCH Models. R Package Version 1.3-4," 2014, available online in May 2015: http://CRAN.R-project.org/package=rugarch.

[32] D. Spinellis, "A Tale of Four Kernels," in *Proceedings of the International Conference on Software Engineering (ICSE 2008)*. Lepzig: ACM, 2008, pp. 381–390.

[33] J. M. Osier, B. Kehoe, C. Support, and Y. Svendsen, "Keeping Track. Managing Messages with GNATS, the GNU Problem Report Management System," 2001, version 4.0-beta1. Available online in June 2014: http://www.gnu.org/software/gnats/doc/4_0_beta1/gnats.pdf.

[34] G. Kuk, "Strategic Interaction and Knowledge Sharing in the KDE Developer Mailing List," *Management Science*, vol. 52, no. 7, pp. 1031–1042, 2006.

[35] A. Silvestrini and D. Veredas, "Temporal Aggregation of Univariate and Multivariate Time Series Models: A Survey," *Journal of Economic Surveys*, vol. 22, no. 3, pp. 458–497, 2008.

[36] C. Izurieta and J. Bieman, "The Evolution of FreeBSD and Linux," in *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering (ISESE 2006)*. Rio de Janeiro: ACM, 2006, pp. 204–211.

[37] N. Jørgensen, "Developer Autonomy in the FreeBSD Open Source Project," *Journal of Management & Governance*, vol. 11, no. 2, pp. 119–128, 2007.

[38] R. F. Engle and T. Bollerslev, "Modelling the Persistence of Conditional Variances," *Econometric Reviews*, vol. 5, no. 1, pp. 1–50, 1986.

[39] D. Kwiatkowski, P. C. Phillips, P. Schmidt, and Y. Shin, "Testing the Null Hypothesis of Stationarity Against the Alternative of a Unit Root: How Sure Are We that Economic Time Series Have a Unit Root?" *Journal of Econometrics*, vol. 54, no. 3, pp. 159–178, 1992.

[40] B. Pfaff, *Analysis of Integrated and Cointegrated Time Series with R*, 2nd ed. New York: Springer-Verlag, 2008.