Empirical Derivations from an Evolving Test Suite

Jukka Ruohonen
University of Southern Denmark
Email: juk@mmmi.sdu.dk

Abhishek Tiwari
University of Southern Denmark
Email: abti@mmmi.sdu.dk

***Abstract*—The paper presents a longitudinal empirical analysis of the automated, continuous, and virtualization-based software test suite of the NetBSD operating system. The longitudinal period observed spans from the initial roll out of the test suite in the early 2010s to late 2025. According to the results, the test suite has grown continuously, currently covering over ten thousand individual test cases. Failed test cases exhibit overall stability, although there have been shorter periods marked with more frequent failures. A similar observation applies to build failures, failures of the test suite to complete, and installation failures, all of which are also captured by the NetBSD's testing framework. Finally, code churn and kernel modifications do not provide longitudinally consistent statistical explanations for the failures. Although some periods exhibit larger effects, including particularly with respect to the kernel modifications, the effects are small on average. Even though only in an exploratory manner, these empirical observations contribute to efforts to draw conclusions from large-scale and evolving software test suites.**

***Index Terms*—software testing, software evolution, test execution monitoring, flaky tests, regression testing, operating systems**

I. Introduction

Operating systems require and benefit from testing like any other software. However, continuous and automatic testing of operating systems has long been a challenge due to the nature of operating systems, including their kernels who deal with hardware and low-level characteristics in general. Despite the overall challenge, progress has been made during the past two decades, including in the open source software (OSS) world. Continuous and automated fuzzing of OSS operating system kernels would be a good example [1]. Also more traditional automated testing frameworks suites have been introduced.

The NetBSD's test suite is based on automated testing framework developed in the early 2010s. Among other things, the framework's benefits include automated runs in a virtual machine [2]. Specifically, the NetBSD project has a continuous system that continuously builds the whole operating system from source code, install the whole operating system to a Qemu-based virtual machine, and runs the individual test cases within the virtual machine installation. The individual test cases cover anything from unit to integration and stress tests.

In addition, the testing frameworks supports different instruction set architectures as well as the NetBSD's own implementations for running kernel code in userland [3]. All in all, the testing framework can be seen as a NetBSD's answer to the common problem about a lack of a universal automation tool for testing operating systems, including those used for embedded devices [4]. Given these introductory points, the test suite provides an excellent case study for examining what can be empirically and longitudinally derived from the test results outputted by the continuous testing framework. To the best of the author's knowledge, no prior work has been done to examine the NetBSD's test suite. Regarding the phrasing about what can be derived, the opening Section II presents three research questions (RQs) for motivating the empirical derivation in relation to existing research. Materials and methods are subsequently described in Section III. Results follow in the next Section IV. A short conclusion and a brief accompanying discussion are presented in the final Section V.

II. Motivation and Related Work

A. Software Evolution

The so-called "laws" of software evolution have been continuously discussed, theorized, and evaluated during the past three or four decades of software engineering research. Although originally these were formulated rather vaguely, thus leaving a wide room for interpretation, the laws' basic message resonates with the discipline's core premises; the sizes of most software projects grow continuously, and, therefore, also complexity is continuously increasing and must be dealt with by continuous modifications, whether maintenance measures or refactoring [5]. The later metaphor of technical debt conveys the same message [6]. For robustly dealing with the accumulating debt, also a test suite should grow continuously; otherwise, refactoring and other maintenance measures may be risky. Although test coverage is not observed, the following simple question is thus fundamental and hence worth asking:

- RQ.1: *Does the NetBSD's test suite indicate continuous growth in terms of individual test cases?*

The second research question is about failure trends:

- RQ.2: *Have the amounts of ❶ failed test cases, ❷ build failures, ❸ failures of the test suite to complete, and ❹ installation failures remained stable over the years?*

Answers to RQ.2 can be used to deduce, even if only tentatively, whether the testing results indicate periods during which the NetBSD project might have had development and maintenance problems. With respect to RQ.2❶, a growth trend instead of longitudinal stability could be seen as a sign that some problems are present. Alternatively, large amounts of failed test cases, or other failures indicated by the test suite, in some particular period could signpost intensified or large-scale restructuring and refactoring or the introduction of large features during which failures are presumably more common.

Although with negative results, a somewhat similar reasoning has been previously used in relation to major FreeBSD releases and volatility of development activity metrics [7]. This reasoning is worth pointing out because FreeBSD has traditionally relied on feature testing and manual testing of releases before their launches [8]. With this point in mind, the NetBSD's testing framework has also improved release engineering [3]. Thus, all in all, also RQ.2 can be linked to existing research.

B. Code Churn and Kernel Code

The third research question seeks to probe whether the longitudinal failure amounts correlate with two software metrics:

- RQ.3: *Can ① code churn and ② kernel modifications prior to test runs explain the ❶ amounts of failed test cases, ❷ whether a build failed, ❸ whether the test suite failed to complete, and ❹ whether an installation failed?*

With respect to RQ.3①, there is a vast literature branch using different code churn metrics for modeling and predicting different software engineering phenomena. The application domains range from bug [9] and vulnerability [10] prediction to fuzzing [1]. Given these domains, it seems sensible to hypothesize that increased churn would also increase the amounts of test case failures and other failures detected by the NetBSD's automated test suite. Regarding RQ.3②, the motivation is even more on the exploratory side. Nevertheless, there are some existing results hinting that kernel modifications could correlate with some particular failure types. For instance, it seems sensible to assume that introducing bugs to kernel code would increase particularly a likelihood that a subsequent installation fails. Empirically, the assumption makes sense because the majority of code changes in operating system projects are either about userland code or kernel code, including in the NetBSD operating system [11]. Thus, also RQ.3 can be connected to existing research albeit only loosely.

Despite the three RQs and their brief motivations, it should be emphasized that the paper is still an exploratory case study whose "primary purpose is to examine a little understood issue or phenomenon to develop preliminary ideas and move toward refined research questions by focusing on the 'what' question" [12, p. 53]. Although there are existing empirical studies on the evolution of test suites [13], [14], as said, the NetBSD's test suite has not been previously examined. The knowledge seems limited also regarding test suites for operating systems in general. With respect to the phrasing about "little understood issue or phenomenon" in the previous quote, RQ.3 is the reference point. To best of the author's knowledge, the question has neither been examined nor answered previously. Thus, even the "preliminary ideas" noted in the quote can provide insights for moving forward.

III. Materials and Methods

A. Data

The dataset was collected from the NetBSD's online archives for the results from the automated test runs [15]. The dataset is also available online for replication and other purposes [16]. The period observed ranges from the first day of August 2011 to the last day of September 2025. Although NetBSD supports multiple instruction set architectures [17], the analysis is restricted to the conventional 32-bit and 64-bit `x86` architectures, as denoted in NetBSD with the `i386` and `amd64` labels. The reason is that the test suite has successfully ran the longest for these two architectures. Although also other architectures are tested with the automated framework, the testing has started later than with `i386` and `amd64`. Furthermore, some of the more exotic architectures have occasionally been broken for relatively long periods of time, meaning that an installation has failed and thus also the test suite has failed to execute. Such periods cause irregularities and other disturbances for the corresponding time series.

In practice, the dataset was assembled by parsing each file in the online archives line by line. As soon further discussed in Subsection III-C, the archival files are provided for each architecture on monthly basis. The following snippet [18] can be used to illustrate the nature of the textual archival data:

```
build: OK with 737188 lines of log, install: OK,
       tests: 10226 passed, 608 skipped,
       87 expected_failure, 2 failed,
       ATF output: raw, xml, html
       new failures: lib/libc/gen/t_sleep/kevent
commit 2025.01.23.12.32.38 christos
       src/tests/lib/libexecinfo/Makefile 1.9
commit 2025.01.23.12.32.38 christos
       src/tests/lib/libexecinfo/t_backtrace_sandbox.c 1.1
build: failed with 646199 lines of log
commit 2025.01.23.12.36.14 christos
       src/distrib/sets/lists/debug/mi 1.460
commit 2025.01.23.12.36.15 christos
       src/distrib/sets/lists/tests/mi 1.1358
build: OK with 737095 lines of log, install: OK,
       tests: 10226 passed, 608 skipped,
       87 expected_failure, 3 failed [...]
```

When starting from the top, it can be seen that a test suite run completed with over ten thousand passed tests. Afterwards, two commits were made to files within the `src/tests` directory where the tests reside. These broke the build. After two subsequent commits were made, the build again succeeded, but the corresponding test suite run indicated one additional test failure compared to the previous successful run. Although only a build failure is shown in the snippet, the test suite also records a failure in case a Qemu-based installation failed. If a build and an installation succeeded, but the test suite failed to run successfully in the virtual machine installation, a further failure is recorded. These failure cases contain everything and anything from timeouts to hangs of the operating system itself.

B. Metrics

The NetBSD's test suite classifies test results into four categories: passed, skipped, failed, and expectedly failed. The skipped category is used for so-called "flaky tests"—tests whose results are non-deterministic [19], as well as for other related reasons. The expected failure category denotes test cases in which a developer has explicitly flagged a failure as an expected result. These cases range from long-lasting bugs to other known outcomes, such as features in standards that have not been implemented. The failed category is used for RQ.2, while an answer to RQ.1 is solicited simply by summing all the outcome categories and then carrying out an aggregation. The

three failure cases noted in the preceding subsection provide the remaining metrics for answering to the question RQ.2. Regarding metrics used in the domain of software testing and their classifications more generally [20], the metrics used are about testing and test case quantities. This characterization is further reinforced by the aggregation soon discussed.

Although no universally accepted definitions are available for code churn, it has typically been measured as source code lines changed, deleted, and added between commits or larger aggregates such as releases [1], [9], [11]. The present work diverges from this tradition by using plain commit counts as a proxy for churn, as has been done also in previous research [21]. With RQ.3① and RQ.3❷ in mind, the example snippet shown earlier would yield a value two in relation to the particular build failure shown in the snippet.

Regarding the kernel modifications, these were identified by commits involving files in the `sys` root directory of the NetBSD's source code tree. Also this operationalization has been previously used [22]. Given that these kernel commits and the total commit counts correlate, the answer to RQ.3② is sought with a dummy variable taking a value zero unless any of the preceding commits before a test run touched files in the `sys` directory in which case the variable's value is one.

C. Aggregation

For RQ.1 and RQ.2, monthly aggregates are used, meaning that the length of the time series is 170. This aggregation frequency is a natural choice in a sense that NetBSD provides separate monthly data archives about the testing results. Regarding aggregation functions, mean and median are used to the amounts of tests and failed tests. For the build failures, test suite failures, and installation failures, monthly sums are used because averaging is not a good choice for these metrics. In other words, a count of build failures in a month, for instance, conveys better information about potential problems than some average amount of build failures recorded during the month.

It is important to further emphasize that RQ.1 and RQ.2❶ are approached by deleting the observations in which a build, the test suite, or an installation failed prior to the aggregation operations. Without the deletion, there would be undefined values in these two two series; no failed test cases, or any other testing results, are recorded under the three failure cases. However, a similar deletion does not obviously make sense for seeking answers to RQ.2❷, RQ.2❸, and RQ.2❹.

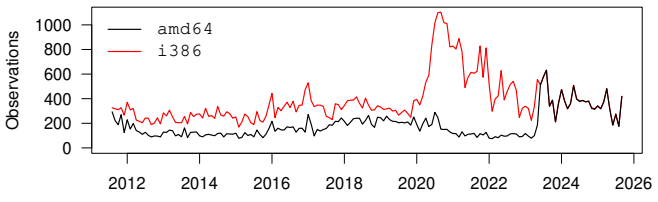

Fig. 1. Monthly Sample Sizes

Regarding RQ.3, the regression analyses soon described in Subsection III-D are computed primarily with unaggregated monthly data. The reason is partially practical and related to the data collection: because NetBSD provides the per-architecture online archival files on rolling monthly basis, it is not straightforward to merge and robustly synchronize the files into annual aggregates or a single large dataset covering the whole period observed. Furthermore, it should be emphasized that the monthly archival files and hence also the monthly unaggregated samples vary across the architectures. The reason is that the virtualization-based test suite is executed at different paces for different architectures. Although the documentation, which may be outdated, says that the test suite is being run circa once per day for `amd64` and circa six times per day for the `i386` architecture [15], Fig. 1 indicates that this scheduling seems to have applied only between circa 2021 and 2024. In other words, when this period is excluded, the sample sizes between `i386` and `amd64` are roughly comparable.

D. Methods

A simple visual inspection is sufficient for answering to the first research question RQ.1. Although formal time series methods and tests, such as those originating from fluctuation processes [23], could be used for answering to RQ.2, a visual interpretation provides the primary means for answering also to this research question. By following recent research [24], the Kruskal-Wallis [25] is briefly also used to check whether the failures differ between two periods from August 2011 to August 2018 and September 2018 to September 2025.

The ordinary least squares (OLS) regression is used for probing a tentative answer to RQ.3❶ and logistic regression (LR) to the remaining RQ.3❷, RQ.3❸, and RQ.3❹. Code churn, as measured by the commit counts, and the dummy variable for kernel modifications are the only two explanatory metrics used. Therefore, the interest is *not* to seek good overall statistical performance but to check whether the two metrics correlate with failure amounts. Regarding the two explanatory metrics, these refer to preceding observations prior to a given test suite run or a failure; the counting starts over from zero once the given run or failure has occurred in the archival files.

As noted in the preceding Subsection III-C, unaggregated monthly data is used for both the OLS and LR regressions, meaning that in total $4 \times 2 \times 170 = 1,360$ individual regression are estimated, given the four different dependent variables, two architectures, and the length of the time series. Therefore, a healthy grain of salt should be taken particularly with statistical significance. Among other things, both model calibration and diagnostics are difficult to do with such a large amount of regressions, a Bonferroni correction might be considered, and so forth and so on. For these reasons, the results are reported by focusing on the regression coefficients from the OLS regressions and the marginal effects from the LR regressions; the latter proxy affects directly upon probabilities. Even with the limitations noted, the monthly regressions have a benefit in that these can reveal whether effects are present in some particular months or longer periods. That is: even in case the effects are modest generally, it could be that some particular months or periods still show strong statistical effects.

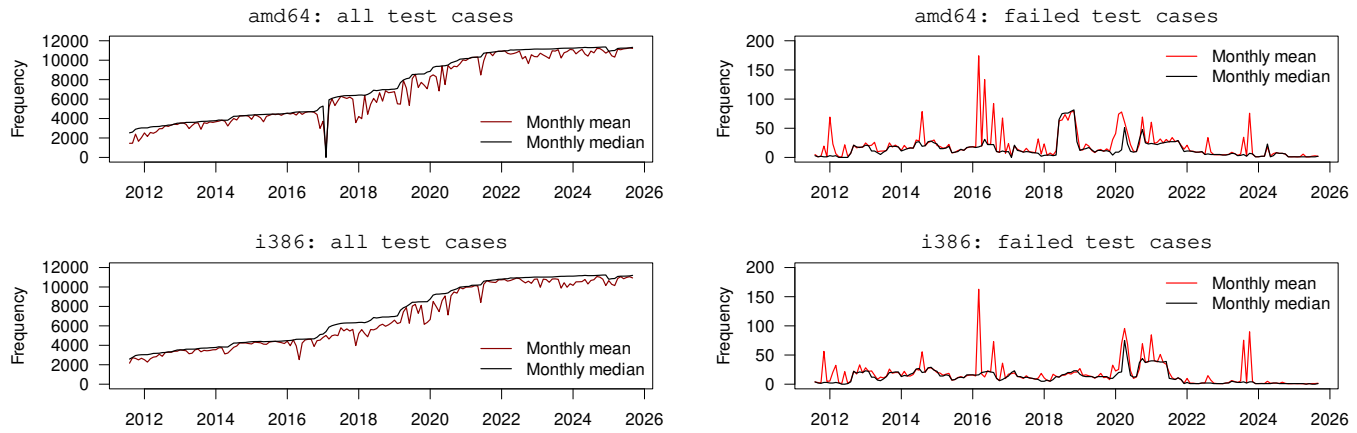

Fig. 2. Test Cases and Failed Test Cases (monthly averages)

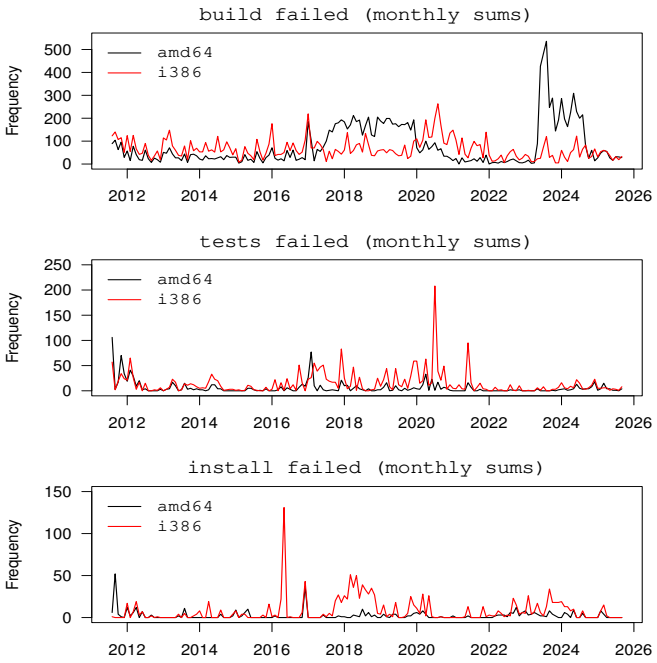

Fig. 3. Build, Test Suite, and Installation Failures (monthly sums)

TABLE I
KRUSKAL-WALLIS NON-PARAMETRIC RANK SUM TESTS

		amd64	i386
Series	Aggregation	p-value	p-value
Failed test cases	Monthly mean	0.095	< 0.001
	Monthly median	0.152	< 0.001
Build failed	Monthly sums	0.153	0.029
Tests failed	Monthly sums	0.080	0.341
Install failed	Monthly sums	0.037	0.090

In addition, brief checks are computed with pooled unaggregated data for the two architectures. The amount of observations are 65,001 and 31,840 for i386 and amd64, respectively. Besides OLS and LR regressions with the pooled unaggregated data, the results for the failed test cases are also checked by estimating a negative binomial regression with a log-link and a first-order autoregression, AR(1), term using an existing implementation [26]. The reason for this additional check is that the pooled time series exhibit pronounced count data characteristics. However, as was noted in Subsection III-C, the pooled estimates suffer from a limitation that the monthly archival files are not properly synchronized. Therefore, a grain of salt is again needed for interpretation.

IV. RESULTS

A. Growth (RQ.1)

The answer to RQ.1 is clear: the test suite has continuously grown—as has most OSS operating systems themselves [27], including NetBSD presumably with them. The answer also places the NetBSD operating system into a distinct cluster of OSS projects with growing test suites [28]. This observation can be seen from the left-hand side plot in Fig. 2, which shows the monthly means and medians of the individual test case amounts. The last observations in the dataset indicate as many as 11,326 test cases for amd64 and 11,198 for the old i386 architecture. Even when keeping in mind that operating systems, including NetBSD, have large code basis, these amounts are large enough to conclude that the automated test framework has extensively and continuously been used in NetBSD throughout the period observed. However, the amount of new test cases added seems to have slowed down or stabilized from circa 2022 onward. Although testing coverage is not observed, as already noted, this observation could be used to contemplate in further work whether a satisfactory coverage level has been reached. Alternatively, it could also be that most of what can be easily tested is already tested.

B. Failures (RQ.2)

The right-hand side plot in Fig. 2 indicates overall long-run stability of failed test cases over time both with noticeable periods during which failures have been more frequent. A similar

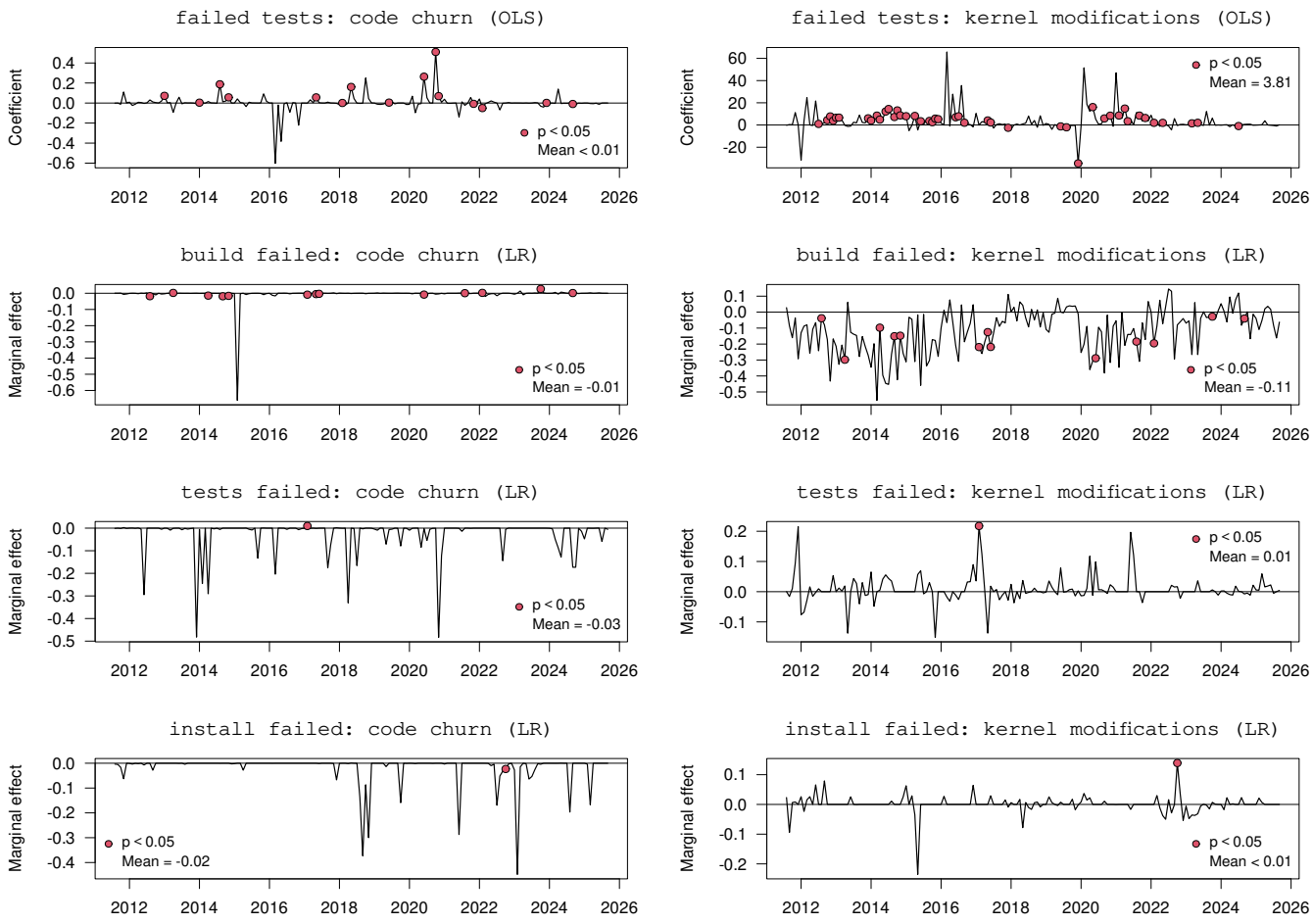

Fig. 4. Regression Results for the `amd64` Architecture (separate monthly estimates; coefficients or marginal effects of the coefficients)

observation can be made about the monthly build failure sums shown in the topmost plot in Fig. 3. Though, the fluctuations in the two time series do not match longitudinally. For the failed test results, the period between circa late-2018 to late-2021 stands out, whereas particularly `amd64` saw many build failures between 2018 and 2020, and then again in 2023 and 2024. Whatever the reasons may be behind these fluctuations, it cannot be concluded that the two time series would be entirely stable. This conclusion is partially supported also by the twofold sample-splitting test results shown in Table I.

If the conventional 95% confidence level is used, the null hypotheses of equal medians between the two periods are rejected for the monthly mean and median series for the failed test cases and the monthly sums for the build failures affecting the `i386` architecture. Interestingly enough, the null hypotheses remain in force for `amd64`, although the `amd64`-specific fluctuations are visually more pronounced. A null hypothesis is also rejected for the installation failures for `amd64`. When taking a look at the three series in Fig. 3, however, a conclusion based on visual inspection seems better. In other words, the test suite and install failures can be concluded to be relatively stable overall despite the fluctuations, whereas a reverse conclusion can be said to apply to the test and build failures. Although reporting of test results can be omitted for brevity, all of the four failure series are still stationary, meaning that particularly their means remain stable over the whole period observed. This point provides a statistical time series ground also for the regression results presented next.

C. Statistical Effects (RQ.3)

The unaggregated monthly regression estimates are summarized in Fig. 4 and Fig. 5 for the `amd64` and `i386` architectures, respectively. In each plot shown in the figures, the effects of the code churn and kernel modification metrics are shown on the y-axes, whereas the x-axes denote the corresponding months used to separately estimate the OLS and LR regressions. When keeping the points made in Subsection III-D in mind, it can be started by concluding that the overall effects are only modest—or even small. This conclusion becomes clear by taking a look at the means of the effects across the regressions, as reported in a legend shown in each plot.

The failed test time series seem to be an exception to the conclusion. In particular, the kernel modification dummy variable shows notable effects upon the unaggregated test

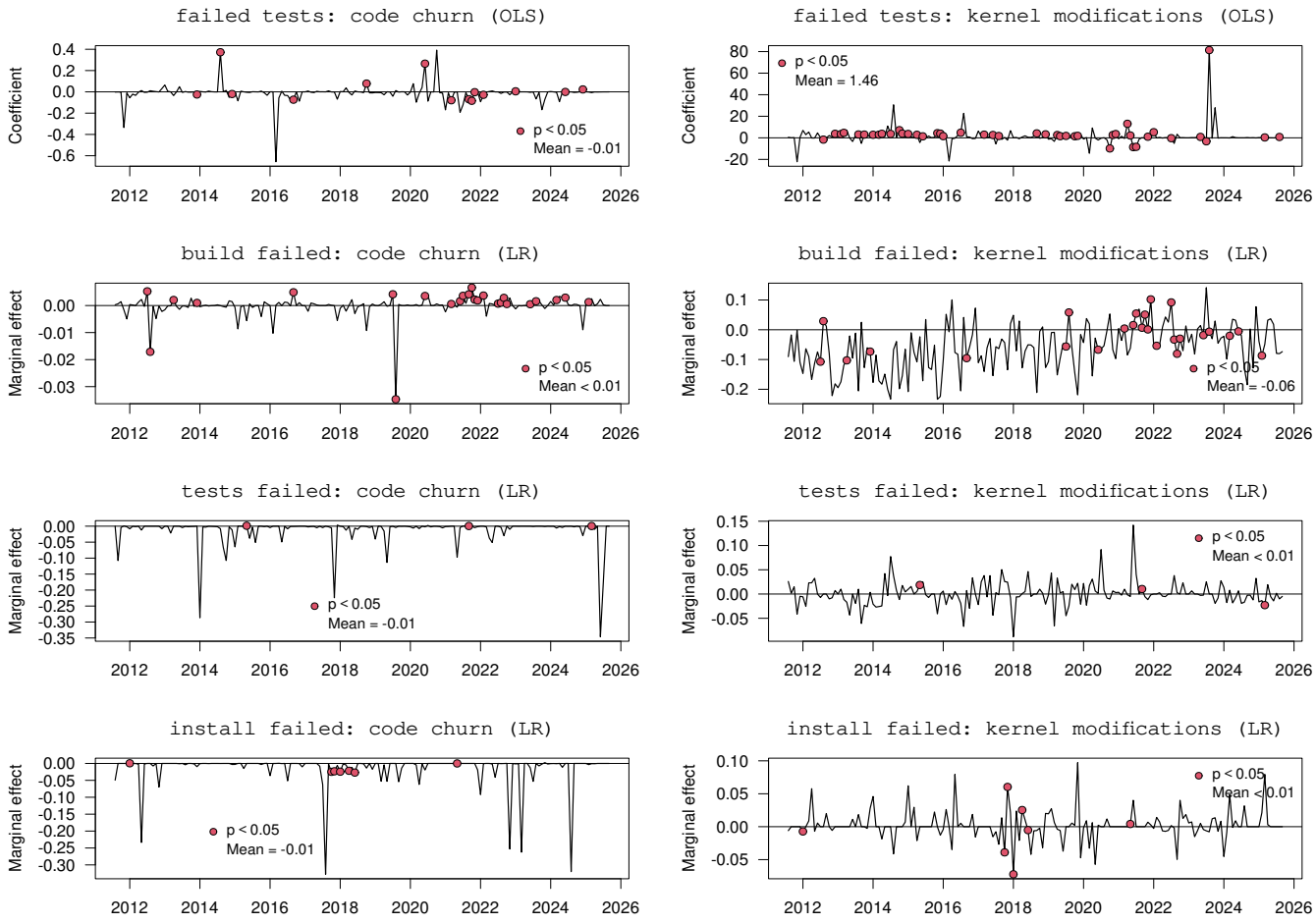

Fig. 5. Regression Results for the `i386` Architecture (separate monthly estimates; coefficients or marginal effects of the coefficients)

failure amounts. When compared to userland modifications, one or more preceding commits having touched files in the `sys` directory tends to increase the amount of failed `amd64` test cases by nearly four tests, all other things being constant. The effect is visible also for `i386`, although, as seen from the top-right plot in Fig. 5, the averaged effect is largely due to a significant effect during a single month in late 2023. When taking a closer look at the individual plots, a similar conclusion applies also more generally: the effects may be small on average, but there are occasionally notable effects during some particular months. Thus, again, the conclusion about fluctuations apply also to the regression estimates.

A further point to make is about the signs of the effects. As seen from the right-hand side plots on the second rows in Figs. 4 and 5, kernel modifications have tended to decrease the amounts of build failures. Although various explanations may be possible, one potential contemplation is that modifying kernel code does not often require modifications to build scripts, configuration files, and related elements. In fact, the test suite itself could be seen to imply frequent build failures because each test case addition requires also modifying bookkeeping material about files present in an installed NetBSD operating system. Furthermore, regarding the signs, it can be also seen and concluded that the code churn metric exhibits negative effects in almost all regressions for the test suite execution and install failures. This point aligns with the previous speculation about different sections in the NetBSD's source code tree causing different failures. Not all failures are common.

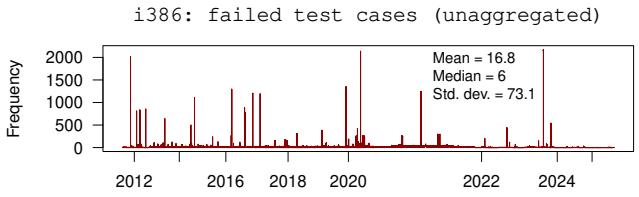

Fig. 6. An Example of Unaggregated Time Series

Finally, to return to the conclusion about modest or even small effects on average, a few remarks can be made about the pooled regression estimates. Before continuing, it should be remarked that the pooled unaggregated data yields irregular time series due to the unequal sample sizes in the

TABLE II
OLS AND LR ESTIMATES WITH POOLED UNAGGREGATED DATA

	Effect	amd64	i386
Code churn:			
Failed test cases	Coef. (OLS)	< 0.001	< 0.001
Build failed	Mar. eff. (LR)	< 0.001	< 0.001
Tests failed	Mar. eff. (LR)	< 0.001	< 0.001
Install failed	Mar. eff. (LR)	< 0.001	< 0.001
Kernel modifications:			
Failed test cases	Coef. (OLS)	5.711	-0.010
Build failed	Mar. eff. (LR)	-0.088	-0.027
Tests failed	Mar. eff. (LR)	0.012	0.012
Install failed	Mar. eff. (LR)	-0.002	-0.007

TABLE III
ANSWERS TO THE RESEARCH QUESTIONS

Question	Answer	Explanation
RQ.1	●	The test suite has continuously grown.
RQ.2❶	◐	Overall, the amounts of failed test cases have been stable, but the time series indicate also fluctuations during which failures have been more frequent.
RQ.2❷	◐	Although overall stability is present, there have also been occasional periods marked with frequent build failures, and these failure periods also differ between the amd64 and i386 architectures.
RQ.2❸	●	There has been overall stability with some short-lived bursts. When compared to the answer to RQ.2❷, the fluctuations have been short and affected particularly i386, possibly due to more frequent runs of the test suite for i386.
RQ.2❹	●	The answer is analogous to the one for RQ.2❸.
RQ.3❶	◐	① Code churn and ② kernel modifications do not have consistent effects upon the four failure cases on average. Only the kernel modifications show notable effects upon test failures. Yet, there are periods during which other effects are visible too.
RQ.3❷	◐	
RQ.3❸	◐	
RQ.3❹	◐	

monthly archival files. If i386 is taken as an example, the fluctuations in the earlier Fig. 1 are seen in a much longer period of observations between 2020 and 2022 in Fig. 6, which also demonstrates the count data characteristics of the pooled data. With these remarks in mind, it can be seen from Table II that the only notable effect belongs to the kernel modifications dummy variable upon the failed test cases in the amd64 architecture. All other effects are more or less close to zero. Also the negative binomial regression noted in Subsection III-D yields coefficients close to zero for both architectures, including for the kernel modifications. The estimated dispersion parameters are as large as 227 and 241 for the two architectures. Furthermore, the AR(1) terms yield coefficients with values 0.784 and 0.928. Thus, the pooled OLS estimates in Table II cannot be seen as robust. All in all, these additional checks can be interpreted to signify that the effects of code churn and kernel modifications are small or even non-existent on average—yet they are also longitudinally inconsistent, meaning that there are still some particular short periods during which effects are visible and likely not noise.

V. CONCLUSION AND DISCUSSION

The answers reached are summarized in Table III within which the symbol ● denotes a positive answer with a reasonable confidence, the symbol ◐ a "partially" positive answer with tentative evidence and interpretation, and the symbol ○ a negative answer to a reasonable degree of evidence and interpretation confidence. This categorization warrants a brief discussion in relation to so-called negative and positive results.

Negative results—understood generally as results that do not support prior expectations—have been praised also in software engineering [29]. However, there are reasons why the concept is poor for describing the results reached and presented. To put aside the issue whether it even makes sense to dichotomously frame results as negative (positive) with respect to some prior positive (negative) expectations, the exploratory nature of the paper prevents characterizing the results as negative. In other words, none of the research questions that were specified in Section II came with solid prior expectations seeking either an empirical confirmation or an empirical falsification. Rather, the many ◐ symbols in Table III would support an alternative term of inclusive results (cf. [30]). This point applies particularly for the answers given to RQ.3; the effects are small or even non-existent on average but the conclusion cannot be generalized to all periods observed. An analogous point applies with respect to the fluctuations affecting the answers to the question RQ.2.

The inconclusive results would provide an opportunity to move beyond the exploratory "what" questions that were noted in Subsection II-B. In other words, it would be worthwhile to understand why there have been fluctuations, what has caused them, and how NetBSD developers themselves perceive them. Likewise, it would be relevant to know why kernel modifications may sometimes, but not always, cause more test cause failures but less build failures, and so forth. Though, there are many "what" questions that would warrant further research too; among other things, the paper did not examine what is actually being tested in NetBSD. The point about potential stabilization or even saturation made in Subsection IV-A would offer a good research question for further work in this regard.

Regarding limitations, the synchronization issues noted in Subsection III-C are a notable threat to internal validity. Also the statistical issues briefly noted in Subsection III-D can be mentioned in this regard. As a case study, external validity too is an inevitable limitation. However—because test suites differ substantially between projects, it seems sensible to expect that generalizability remains an issue also in further work. In contrast, the synchronization issues would seem more plausible to address in further work. It should be noted that the issue is not really about merging the monthly archival files into a single file or a database entry—such a task seems almost trivial. Rather, the issue is more about how to do sensible time series analysis when data is irregular and should be broken into smaller subsets. The question could be extended to bisecting of failures that is a common task and functionality in automated testing frameworks [1]. Also the NetBSD's testing framework does bisecting to buggy commits but it is unclear how robustly.

References

[1] J. Ruohonen and K. Rindell, "Empirical Notes on the Interaction Between Continuous Kernel Fuzzing and Development," in *Proceedings of the IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW 2019)*. Berlin: IEEE, 2019, pp. 276–281.

[2] A. Kantee, "Testing NetBSD: Easy Does It," 2010, NetBSD Blog, available online in October 2025: https://blog.netbsd.org/tnf/entry/testing_netbsd_easy_does_it.

[3] M. Husemann, "Testing NetBSD Automagically," 2011, EuroBSDCon, available online in October 2025: https://www.netbsd.org/gallery/presentations/martin/eurobsdcon2011/testing_netbsd_automagically.pdf.

[4] S. M. A. Shah, D. Sundmark, B. Lindström, and S. F. Andler, "Robustness Testing of Embedded Software Systems: An Industrial Interview Study," *IEEE Access*, vol. 4, 2016.

[5] J. Ruohonen, S. Hyrynsalmi, and V. Leppänen, "Time Series Trends in Software Evolution," *Journal of Software: Evolution and Process*, vol. 27, no. 2, pp. 990–1015, 2015.

[6] C. A. Siebra, A. Cavalcanti, F. Q. Silva, A. L. Santos, and T. B. Gouveia, "Applying Metrics to Identify and Monitor Technical Debt Items During Software Evolution," in *Proceedings of the IEEE International Symposium on Software Reliability Engineering Workshops (ISSRE Wksp 2014)*. Naples: IEEE, 2014, pp. 92–95.

[7] J. Ruohonen, S. Hyrynsalmi, and V. Leppänen, "Software Evolution and Time Series Volatility: An Empirical Exploration," in *Proceedings of the 14th International Workshop on Principles of Software Evolution (IWPSE 2015)*. Bergamo: ACM, 2015, pp. 56–65.

[8] T. Dinh-Trong and J. M. Bieman, "Open Source Software Development: A Case Study of FreeBSD," in *Proceedings of the 10th International Symposium on Software Metrics*, 2004, pp. 96–105.

[9] E. Giger, M. Pinzger, and H. C. Gall, "Comparing Fine-Grained Surce Code Changes and Code Churn for Bug Prediction," in *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR 2011)*. Waikiki, Honolulu: ACM, 2011, pp. 83–92.

[10] C. Theisen and L. Williams, "Better Together: Comparing Vulnerability Prediction Models," *Information and Software Technology*, vol. 119, p. 106204, 2020.

[11] G. Kudrjavets, J. Thomas, N. Nagappan, and A. Rastogi, "Is Kernel Code Different From Non-Kernel Code? A Case Study of BSD Family Operating Systems," in *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME 2022)*, Limassol, 2022, pp. 211–222.

[12] V. B. Kampenes, B. Anda, and T. Dybøa, "Flexibility in Research Designs in Empirical Software Engineering," in *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE 2008)*. Swindon: BCS Learning & Development Ltd., 2008, pp. 49–57.

[13] W. Aljedaani and Y. Javed, "Empirical Study of Software Test Suite Evolution," in *Proceedings of the 6th Conference on Data Science and Machine Learning Applications (CDMA 2020)*. Riyadh: IEEE, 2020, pp. 87–93.

[14] L. S. Pinto, S. Sinha, and A. Orso, "Understanding Myths and Realities of Test-Suite Edevolution," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE 2012)*. Cary North Carolina: ACM, 2012, pp. 1–11.

[15] The NetBSD Foundation, Inc., "Offical Test Suites," 2025, available online in October 2025: https://releng.netbsd.org/test-results.html.

[16] J. Ruohonen and A. Tiwari, "A Dataset for a Paper Entitled "Empirical Derivations from an Evolving Test Suite"," 2025, Zenodo, available online in October 2025: https://doi.org/10.5281/zenodo.17461883.

[17] The NetBSD Foundation, Inc., "Platforms Supported by NetBSD," 2025, available online in October 2025: https://wiki.netbsd.org/ports/.

[18] ——, "Offical Test Suites (`amd64` for January 2025)," 2025, available online in October 2025: https://releng.netbsd.org/b5reports/amd64/commits-2025.01.html.

[19] G. Pinto, B. Miranda, S. Dissanayake, M. d'Amorim, C. Treude, and A. Bertolino, "What Is the Vocabulary of Flaky Tests?" in *Proceedings of the 17th International Conference on Mining Software Repositories (MSR 2020)*. Seoul: ACM, 2020, pp. 492–502.

[20] F. Witte, *Metrics for Test Reporting: Analysis and Reporting for Effective Test Management*. Wiesbaden: Springer, 2024.

[21] G. Kudrjavets, N. Nagappan, and A. Rastogi, "Are We Speeding Up or Slowing Down? On Temporal Aspects of Code Velocity," in *Proceedings of the IEEE/ACM 20th International Conference on Mining Software Repositories (MSR 2023)*, Melbourne, 2023, pp. 267–271.

[22] G. Kudrjavets, N. Nagappan, and T. Ball, "Assessing the Relationship between Software Assertions and Faults: An Empirical Investigation," in *Proceedings of the 17th International Symposium on Software Reliability Engineering (ISSRE 2006)*, Raleigh, 2006, pp. 204–212.

[23] J. Ruohonen, "A Time Series Analysis of Assertions in the Linux Kernel," in *Proceedings of the 37th International Conference on Testing Software and Systems (ICTSS 2025), Lecture Notes in Computer Science (Volume 16107)*, S. Bonfanti and G. A. Papadopoulos, Eds. Limassol: Springer, 2026, pp. 3–15.

[24] ——, "Has Complexity of EU Law Increased?" 2025, archived manuscript, available online in October 2025: https://doi.org/10.31235/osf.io/82uan_v1.

[25] W. H. Kruskal and W. A. Wallis, "Use of Ranks in One-Criterion Variance Analysis," *Journal of the American Statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.

[26] T. Liboschik, K. Fokianos, and R. Fried, "tscount: An R Package for Analysis of Count Time Series Following Generalized Linear Models," *Journal of Statistical Software*, vol. 82, no. 5, pp. 1–51, 2017.

[27] Y. Zhao, C. Li, Z. Chen, and Z. Ding, "Dissecting Code Features: An Evolutionary Analysis of Kernel Versus Nonkernel Code in Operating Systems," *Journal of Software: Evolution and Process*, vol. 37, no. 1, p. e2752, 2025.

[28] C. Miranda, G. Avelino, and P. S. Neto, "Test Co-Evolution in Software Projects: A Large-Scale Empirical Study," *Journal of Software: Evolution and Process*, vol. 37, no. 7, p. e70035, 2025.

[29] R. F. Paige, J. Cabot, and N. A. Ernst, "Foreword to the Special Section on Negative Results in Software Engineering," *Empirical Software Engineering*, vol. 22, pp. 2453–2456, 2017.

[30] A. Borji, "Negative Results in Computer Vision: A Perspective," *Image and Vision Computing*, vol. 69, pp. 1–8, 2018.