Effects of an Economic Approach for Test Case Selection and Reduction for a Large Industrial Project

Thomas Bach
Institute of Computer Science
Heidelberg University
69120 Heidelberg, Germany
thomas.bach@stud.uni-heidelberg.de

Ralf Pannemans
SAP SE
Dietmar-Hopp-Allee 16
69190 Walldorf, Germany
ralf.pannemans@sap.com

Sascha Schwedes
SAP SE
Dietmar-Hopp-Allee 16
69190 Walldorf, Germany
sascha.schwedes@sap.com

***Abstract*—Extensive testing in large projects can lead to tremendous test costs with superlinear growth over time. Researchers have proposed several techniques to tackle this problem. However, the practical effects of these techniques on the asymptotic behaviour of test costs growth in large industrial software projects remains poorly characterized.**

We introduce and analyse a fixed time budget for test executions for SAP HANA, a large industrial project. Our approach assigns a global fixed time budget to several components. Each component can only execute tests within its budget, which can change only by transfers from other components. This limits the number of test executions for each test run to a constant, thus reducing the asymptotic growth of test costs.

Budget transfers and test optimizations adhere to balances between value and costs, thus creating an economic environment for test case selection and reduction. Specifically, this creates incentives to remove unnecessary tests and to optimize test execution times.

For SAP HANA, our approach leads to effective test case selection and reduction, and reduces test execution times by 105 years in four months with a negligible effect on quality. The trade-off between runtime savings and failure detection is 1.83 years/failure.

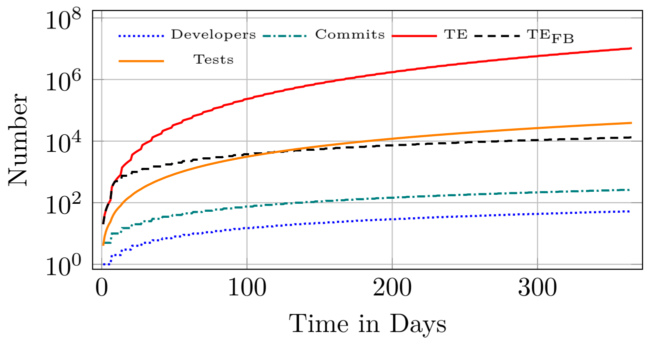

Figure 1. Model of test execution growth in logarithmic scale. The number of developers ($Developers$) increases by one every seven days. Each developer increases the number of existing tests ($Tests$) by four per day and the number of commits ($Commits$) by five code changes per day, each initiating a test run. The number of test executions (TE) is equal to $Tests * Commits$ and grows superlinearly over time due to the linear increases of $Tests$ and $Commits$. A fixed time budget for test executions effectively limits the number of executed tests to a constant c. Therefore, the number of test executions with a fixed budget (TE_{FB}) is equal to $c * Commits$ and grows linearly over time.

I. Introduction

Software testing is a fundamental part of quality assurance for software projects. Agile development processes, continuous integration and short release cycles require frequent automatic test runs to reduce the number of defects and to prevent regressions during development.

Running automated software tests generates costs. Each test run requires CPU time and CPU time translates to hardware costs. The CPU time depends on the test runtime, i.e. the duration between start of a test run until all test executions finished and the results are processed. In addition to the hardware costs, large test runtimes increase waiting times for developers and can decrease productivity. For the rest of the paper, we assume that test runtime correlates with test executions and use it as a surrogate.

For small projects, the overhead in terms of test runtime is rather small. Local test executions on developers' workstations and frequent test runs on central machines (e.g. for each commit) are common practice in smaller projects. Due to the project size, the waiting times are acceptable.

For large projects, the overhead in terms of test runtime grows significantly. Parallel test executions on a hardware pool can reduce test runtimes. However, hardware extensions are limited by resources in practice and cannot sustainably solve the superlinear increase in test runtime.

Test runtimes increase superlinearly due to the typical characteristics of large software projects that affect the number of test executions TE. TE depends on two factors: 1) on the number of tests T that are executed, and 2) on the frequency F the tests are executed. T increases linearly over time because developers add more tests but rarely delete tests. F increases linearly over time because a large successful project has an increasing number of developers who make changes to the central source code repository. Each change initiates a new test run and thus increases F. The multiplication of the two linear factors T and F lead to a superlinear increase for TE over time. The model of Figure 1 demonstrates this superlinear increase of TE. After one year, 53 developers would have produced 39 004 tests, and create 265 code changes per day, resulting in over 10 million test executions per day.

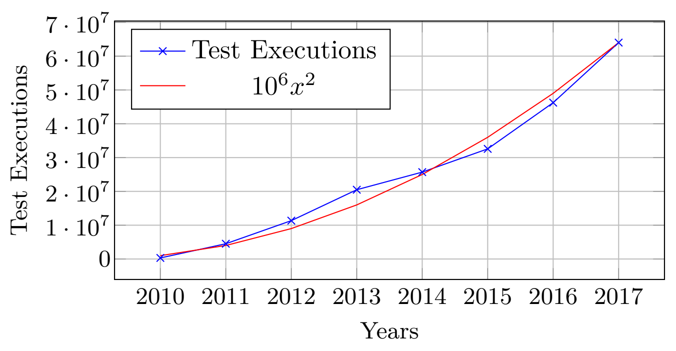

Figure 2. SAP HANA test executions with a quadratic curve fit. Logging contains all centralized automated test executions since 2010.

In addition to this general issue, there are at least two other aspects with a considerable effect on test costs. Long term projects as SAP HANA support multiple product versions for their customers. The support of multiple product versions requires parallel development and test executions due to differences in features and code state. The parallel test executions for multiple product versions lead to an additional constant or potential linear factor of test executions. Non-deterministic tests, often called flaky tests [1], are a second aspect with a considerable effect on test costs in large projects. Flaky tests increase costs for all test stages due additional inspections and reruns for misleading test results.

Overall, we expect that test executions and test costs increase superlinearly in large projects. For SAP HANA, Figure 2 confirms the superlinear growth for test executions between 2010 and 2017. Section IV highlights other works that confirm the superlinear growth.

The growing test costs problem has been addressed by researchers and practitioners for decades: 'Regression testing can be expensive, and the need for cost-effective techniques has helped it emerge as one of the most extensively researched areas in testing over the past two decades.'[2] The approaches to tackle this problem can be categorized in three groups: test case prioritization (TCP) and selection (TCS), and test suite reduction (TSR).

TCP techniques reorder the execution order of tests to maximize some objective function. For example, tests that detect failures should run first. In the case of a detected failure, all subsequent tests could be skipped until the failure is fixed. Typically, most test runs finish without a failure, therefore the potential cost savings are rather low. TCS techniques select a subset of tests for each test run. For example, only test cases relevant to some objective function are selected for a test run. TSR techniques shrink the test suite (i.e. test cases are removed). Therefore, the potential cost savings for TCS and TSR techniques are typically higher compared to TCS, but TCS and TSR techniques have the potential disadvantage of a loss of testing quality because not all tests are executed compared to the full test suite.

For the development of SAP HANA, SAP adopted a fixed time budget for test executions to reduce testing costs. The fixed time budget implements TCS and TSR techniques. Periodic test runs with all tests in later testing stages prevent the potential loss of quality. We analyse the first effects of this approach after four months. The contributions of this paper are as follows:

- A fixed time budget for test executions can be an effective approach for the problem of superlinear test costs growth over time in a large software project.
- A fixed time budget for test executions can create incentives to improve test runtimes, scope, and priority based on economic decisions of value and costs.

Section II introduces the testing environment of SAP HANA and our fixed test budget approach. We describe our approach in detail in Section II-B. Section III contains our research questions, results and discussions. Section IV briefly highlights related work. We conclude in Section V.

II. Testing at SAP HANA and an Economic Approach for Test Case Selection / Reduction

We introduce the testing process of SAP HANA, the application in focus of our analysis, a high-performance, parallel in-memory database management system developed by SAP [3]. Previous work covers details of the testing process and related problems [4]. Due to confidentiality reasons, we cannot disclose all project details. We further describe the fixed time budget for testing.

A. Testing of SAP HANA

Since many customers use SAP HANA in business-critical scenarios, quality assurance of SAP HANA is of paramount importance. SAP ensures this requirement via extensive testing in all development and release stages.

Testing of SAP HANA contains several stages as shown by Figure 3 and involves a *Continuous Build* (CB) environment. At the first stage, developers execute tests during software development on their own workstation. This can include manual tests, a subset of test suites from later testing stages or developer specific test suites. The second stage incorporates the CB environment to automatically execute integration tests within the scope of a component.

Stage three includes the integration and regression testing for all SAP HANA components and causes a majority of the testing costs because of the superlinear cost increase explained in Section I. Figure 2 shows the current trend of test executions for SAP HANA. Stage three implements pre-commit testing, i.e., a software change is only accepted and merged into the repository if it passes all tests.

Within testing stage four, the CB system runs different combinations and configurations of tests (test profiles) after code changes have been submitted to the repository. The execution frequency for these profiles reaches from several times a day to once per release. Stage four includes long running tests, randomized and stress testing, or recovery testing like out of memory or crash tests. The test profile

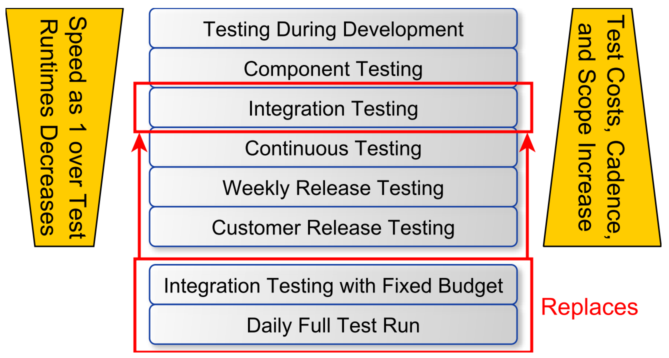
Speed as 1 over Test Runtimes Decreases
Testing During Development
Component Testing
Integration Testing
Continuous Testing
Weekly Release Testing
Customer Release Testing
Integration Testing with Fixed Budget
Daily Full Test Run
Replaces
Test Costs, Cadence, and Scope Increase

Figure 3. Simplified summary of test stages for SAP HANA. For the implementation of our fixed test budget approach, we replace the (full) integration testing stage with two new stages. The two new stages consist of a test run with a fixed time budget for each code change and a daily full test run. The fixed time budget for test executions reduces the costs growth at the integration testing stage. The daily test run ensures the same quality level as before the replacement.

for releases contains over 900 000 single test cases and would run for 23 days if executed sequentially on an average server with 40 CPU cores, 3 GHz frequency, and 256 GiB of RAM. Although SAP utilises a large hardware pool of test servers in parallel mode, the test runtime is rather large. Within the test suites of a database, it is not uncommon to have long running, complex and distributed queries to test database functionality. Performance and load tests contribute to long running tests as well.

Whereas the stages one to four focus on SAP HANA in an isolated environment, stage five and six are used for end-to-end testing in the context of an end user application. These stages contain automatic, but also manual tests that are performed by other teams within the company and thus decrease testing biases.

B. *Fixed Test Budget Approach*

To limit the test runtime of the integration stage, we introduced a fixed runtime budget for tests. More specifically, each component receives an own test runtime budget. Within the budget, the component team can select the tests to run by their own criteria if the cumulative runtime fits into the budget. Time budget can be shifted between components to cope with changing priorities and quality requirements within SAP HANA development. These budget redistributions allow the component team members to focus resources in areas of special interest.

There were several additional expectations based on the incentives given by the optimization of value and costs:

- Tests with low bug finding abilities will move to later testing stages (by manual or automatic techniques).
- Execution times for tests will be analysed and optimised, especially for long running and distributed tests.
- Test scope will improve. A test suite for a component should only include necessary tests for the quality assurance of this component. This could require modifications of the test suite or the component structure.

The implementation of our fixed time budget for test executions requires the determination of an initial total budget and the distribution of this budget to all components. The analysis of several approaches revealed multiple common drawbacks. The usage of empirical data about test executions requires the determination of an appropriate period of observation. A long period would not include recent changes, a short period would not include long term trends and a weighted mix would be rather complex to understand by all developers. In addition, test ownership is not well defined if multiple components use the same tests in their component test suites.

Therefore, the initial time budget distribution depends on the number of developers working per component. The number of developers provides a rough estimation of testing requirements. This is a simple model that can be communicated without further explanations. After the initial distribution, the possibility to transfer test budgets between different components allows a flexible redistribution of time budgets.

The component teams were responsible for the process of test selection and reduction to fulfil the test budget constraint for their respective component. To support the component team members with test case selection and test suite reduction, we created an extensive analysis of runtime and failure rates for the tests contained in the integration stage over the former half year. For example, we found that 17 % of all tests within the old integration stage had a failure rate less than 0.01 % and 11 % had no failure at all for the observation time. The criteria for test selection and reduction which were used by the component teams were not evaluated. We expect a mix of domain knowledge and empirical data. A future study could investigate if these criteria could be utilized for an automated approach.

The introduction of a fixed time budget for integration tests implies a possible reduction of test quality because fewer tests are run. To counter this effect and ensure a consistent quality level for further testing stages, a daily test run contains the original set of integration tests. The daily test run effectively moves the complete test run from pre-commit to post-commit testing.

Altogether, the introduction of the fixed time budget for the integration test stage changes the asymptotic behaviour of the number of test executions TE over time. Before the introduction, TE increased superlinearly over time, as discussed in Section I. After the introduction, TE increases only linearly over time on both new integration test stages. For the integration testing with fixed budget, the test budget limits the maximum runtime of tests. TE depends on a fixed maximum runtime (i.e., a constant) and a variable number of commits over time, resulting in a linear growth for TE over time. For the daily full test run, the frequency is fixed. TE depends on a variable number of tests over time and a fixed execution frequency (i.e., a constant), resulting in a linear growth of TE over time.

III. Results

We investigated the following research questions in the context of the large-scale software project SAP HANA:

RQ1: Does the distribution of the fixed test budget for test executions change over time?

RQ2: How did runtimes for integration tests change after the introduction of the fixed time budget?

RQ3: How is the testing quality affected by the introduction of the fixed test budget for test executions in terms of failures that pass the reduced test suite, but appear in the full integration test suite?

RQ1 analyses whether the possibility to redistribute test budget between components was used by the component teams. This would indicate whether the economic aspect of benefits and costs lead to re-prioritization of testing efforts. RQ2 investigates the direct effects on developer processes and hardware costs. This research question analyses whether test execution times were reduced, one of the main goals for the introduction of the fixed time budget. RQ3 analysis the impact of the fixed time budget on the quality of code change testing. A significant increase in number of defects in later testing stages could imply that the fixed test budget for code changes has a strong negative impact on testing quality. This would question the usefulness of the approach. On the other hand, a low increase in defects in later testing stages or none would indicate that the trade-off between test runtimes and quality is acceptable. In addition, we can combine runtime savings and number of failures at later test stages to quantify the trade-off between runtime and failures in later test stages in terms of test runtime saved per failure.

A. *RQ1 Budget Redistribution*

Figure 4 visualizes how the time budget of each component changed between $t0$ (introduction of the fixed time budget for test executions, 2017-08) and $t1$ (2017-12). Unfortunately, for budget redistributions, we do not have the source and target components for all changes. Therefore, we cannot show an alluvial diagram to illustrate the fine-grained changes. We rescaled the unchanged budget part, because it does not provide further information. The percentage of the unchanged part differs in $t0$ and $t1$ due to an increase in the total time budget. The total time budget increased due to readjustments after the initial introduction. In total, the time budget increased for 18 components and decreased for 27 components between $t0$ and $t1$.

Figure 4 shows that budgets transfers occur in practice to optimize benefits and costs. This is an indicator that the expected economic effect exists in practice. Further investigations of single cases show that there are multiple reasons. Due to space and confidentiality reasons, we list only some of them:

- Components were split to narrow the test scope. This increases the test efficiency as tests are selected closer to the scope of the tested component.

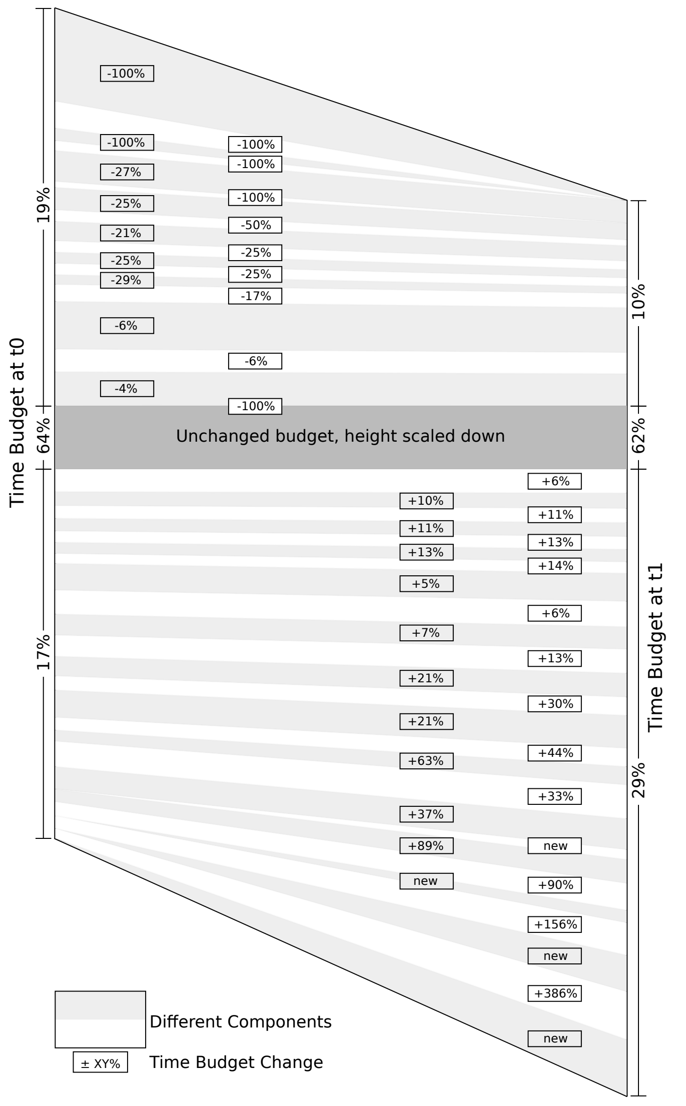

Figure 4. Changes in time budgets for each component over time from $t0$ (introduction of the time budget, 2017-08) to $t1$ (2017-12).

- Required test runtimes were further analysed by component team members and free test budget was reallocated to other components.
- Test runtimes were improved. Long running tests were identified, analysed and improved or unecessary tests were removed.
- Test placement was re-evaluated. Especially long running tests were reallocated to later stages of the testing process with decreased run frequency.

Overall, the distribution of the fixed test budget to components changed over time and the possibility to reallocate test budget is used. Our analysis indicates that this can lead to positive effects on the quality of the test suite due to an incentive to maintain the test suite from a cost and benefit perspective.

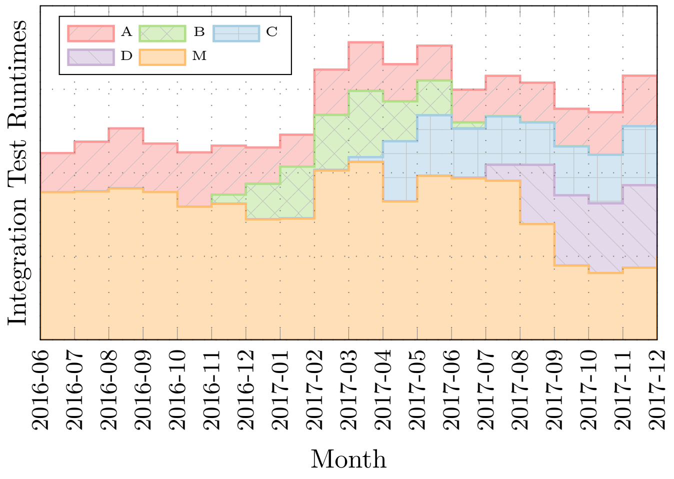

Figure 5. Stacked area chart with test runtimes for product lines M (main), A, B, C, D. The test runtimes for M decreased in August 2017 due to the introduction of the fixed time budget for test executions.

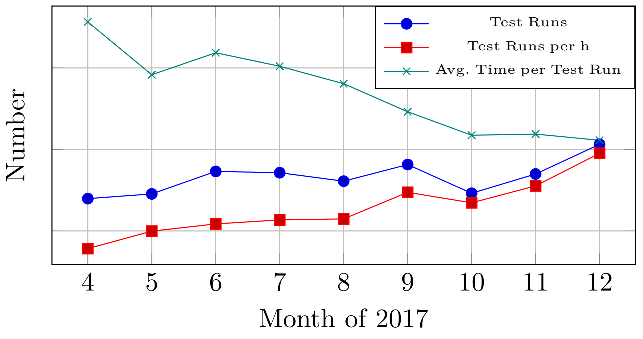

Figure 6. Integration test runtimes and runs over time. Test runtime decreases and test runs increases and therefore test runs per hour increases. Absolute numbers not shown due to confidentiality reasons.

B. RQ2 Changes in Test Runtimes

Within the testing process of SAP HANA (Section II-A), the fixed test budget affects only integration testing for the main product line. Other integration stages have different policies for test selection. We analyse the test cost changes in terms of test runtimes for the integration testing to the main product line and for all integration testing.

The fixed test budget defines the maximum threshold of execution time for integration tests to the main product line. At the point of the introduction t_0 (August 2017), the difference between budget and actual runtime was a factor of 4. After t_0, the actual runtimes are nearly equal to the budget with some deviation. Deviations occur because of re-adjustments and test runtime fluctuations due to variations in infrastructure workload. Figure 5 shows the runtime statistics for multiple product lines. M represents the main product line. Figure 5 shows that the total runtime for M decreased from August to September by a factor of 2. Based on further analysis, we conclude that the reasons for the smaller than expected reduction are test overheads like compile times, binary redistribution, database install and setup times, and test setup times.

Figure 5 shows that the fixed time budget for test executions saved hardware resources, but the figure also shows that additional product versions counterbalance the savings. As a result of this study, SAP engineers currently prepare the introduction of fixed time budgets or similar approaches for other product lines. It is unclear whether other product lines would require the same testing budget as the main product line or if a reduced budget would be sufficient for other product lines.

The overall average runtime for all integration tests exhibits a more complex pattern. Figure 6 visualizes the state before and after the introduction of the fixed test budget. Unfortunately, there is no data logged for the time before 2017 and we cannot analyse long term trends. Due to confidentiality reasons, we cannot provide the exact numbers, Figure 6 only shows the scaled data. Figure 6 shows that the average test runtime R_{avg} decreased after the introduction of the fixed test budget in August 2017. This leads to an increase in test runs TR, which is explained by hardware limitations. TR multiplied with R_{avg} cannot exceed the capacities of the existing hardware. Therefore, if the product is fixed, TR must increase if R_{avg} decreases. The increase in TR and decrease in R_{avg} result in a increase by a factor two for the quotient test runs per hour which represents efficiency.

Overall, Figure 5 and Figure 6 indicate that the fixed test budget has a positive effect on test runtimes and test runtime efficiency. It is unclear whether the current trend continues and if yes, why. We would expect that after the introduction of the test budget, there would be a sharp decline of a factor 4 in the test runtime which would then stay nearly constant. Our data shows that this factor was not fully reached. The ongoing changes could indicate that component teams are continuously improving their tests and therefore optimizing their test budgets.

C. RQ3 Changes in Quality

We analysed all internal bug reports for valid test failures and counted them over time (this excludes flaky test results). Figure 7 shows the results. Statistically, there are in average 3.20 failures per week. This number is remarkable low for a large project as SAP HANA.

In addition, we compare the results from Section III-B and Section III-C to quantify the trade-off between runtime savings and failures in later test stages in terms of test runtime saved per failure. We estimate runtime savings R_s by interpolation of test runtimes without the fixed time budget R_i and subtraction of the actual runtime R_a. In Figure 5, the last known test runtime R before the introduction of the fixed test budget is shown by the datapoint 2017-08 for M. We multiply R by 5 for 5 months. We can determine R_a by the sum over all actual test runtimes in M for the last 5 months. Now, we can calculate the runtime savings $R_s = R_i - R_a$.

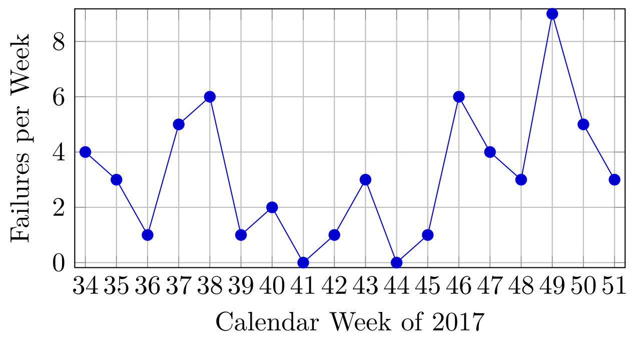

Figure 7. Failures undetected by the reduced test suite but within the full test run. Aggregated by calendar week due to the different behaviour of weekdays and weekends. Average: 3.20 failures per week

Our calculation leads to a result of 104.50 years, which implies that SAP saved 105 years of test execution time due to the fixed time budget. With the average of 3.20 failures per week indicated by Figure 7, we can calculate a quotient of 1.83 years/failure. This implies that SAP traded in average a test runtime of 1.83 years against one additional failure in a later test stage. This number only provides an approximation of the trade-off, and does not necessarily reflect the reality precisely due to the interpolation and the low number of data points.

IV. Related Work

Due to size constraints, we briefly highlight related literature. Yoo et al. provide a survey about test case selection and test suite reduction [5]. A recent systematic literature review about test case selection from by Kazami et al. highlights different trends and results in this area [6]. Blondeau et al. analyse test case selection within several industrial projects [7]. Their work focuses on change based testing, our investigated approach does not depend on dependencies between changes and tests.

Our approach shares characteristics with risk-based testing. Although, as explained in Section II-B, we mitigate risks by a daily full test run. Felderer and Schieferdecker provide a taxonomy of risk-based testing [8]. Risk-based test planning subsumes techniques that select or prioritise test cases based on a risk analysis of their costs and value.

Memon et al. study the same problem of polynomial growth for test executions at Google [9]. The approach used by Google utilizes a manual dependency list for each product to collect all required test runs for a source code change. The test framework collects these required test runs over a period of four hours and runs each required test only once. An automatic backward analysis identifies failure-introducing commits. This approach reduces the linear growth of test executions by commit frequency to a constant factor due to the fixed number of executions per day. However, our fixed test budget creates in addition incentives for developers to reduce and improve existing test suites. Our multi-stage separation enables faster individual feedback times.

V. Conclusions

We described the fixed time budget for testing approach adapted by SAP to limit the test execution growth. We analysed the initial effects on runtimes and quality in terms of failures that pass the reduced test suite, and we analysed the economic effects. Our analysis indicates that there are positive effects on runtime and test suite efficiency, while the negative effects on quality are low.

Although the current observations show possible trends, we cannot conclude statistical significance due to limited data. Further work would require an observation time of at least one year to improve the statistical significance. One year would cover periodic variations, which can be observed for example at the end of the year due to vacations or directly before and after releases of new major software versions.

Based on internal discussions, the results of our analysis are plausible and the impact on test runtimes is and failure detection is reasonable. Surveys or questionnaires with developers would allow to further analyse the impact of the fixed time budget for testing on the individual developer. These findings could support further refinements of our approach. As a direct consequence of this work, the fixed time budget for test executions will be introduced for parallel product lines, as explained and reasoned in Section III-B.

References

[1] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov, "An empirical analysis of flaky tests," in *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014. New York, NY, USA: ACM, 2014, pp. 643–653. (Cited from: I)

[2] A. Orso and G. Rothermel, "Software Testing: A Research Travelogue (2000–2014)," in *Proceedings of the on Future of Software Engineering*, ser. FOSE 2014. New York, NY, USA: ACM, 2014, pp. 117–132. (Cited from: I)

[3] F. Färber, S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner, "SAP HANA database: Data management for modern business applications," *SIGMOD Rec.*, vol. 40, no. 4, pp. 45–51, Jan. 2012. (Cited from: II)

[4] T. Bach, A. Andrzejak, and R. Pannemans, "Coverage-based reduction of test execution time: Lessons from a very large industrial project," in *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, March 2017, pp. 3–12. (Cited from: II)

[5] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Softw. Test. Verif. Reliab.*, vol. 22, no. 2, pp. 67–120, Mar. 2012. (Cited from: IV)

[6] R. Kazmi, D. N. A. Jawawi, R. Mohamad, and I. Ghani, "Effective regression test case selection: A systematic literature review," *ACM Comput. Surv.*, vol. 50, no. 2, pp. 29:1–29:32, May 2017. (Cited from: IV)

[7] V. Blondeau, A. Etien, N. Anquetil, S. Cresson, P. Croisy, and S. Ducasse, "Test case selection in industry: An analysis of issues related to static approaches," *Software Quality Journal*, vol. 25, no. 4, pp. 1203–1237, Dec. 2017. (Cited from: IV)

[8] M. Felderer and I. Schieferdecker, "A taxonomy of risk-based testing," *International Journal on Software Tools for Technology Transfer*, vol. 16, no. 5, pp. 559–568, Oct 2014. (Cited from: IV)

[9] A. Memon, B. Nguyen, E. Nickell, J. Micco, S. Dhanda, R. Siemborski, and Z. Gao, "Taming Google-scale continuous testing," in *ICSE '17:Proceedings of the 39th International Conference on Software Engineering*, 2017. (Cited from: IV)