Decision Making and Visualizations Based on Test Results

Per Erik Strandberg
Westermo R & D AB, Sweden
per.strandberg@westermo.se

Wasif Afzal
Mälardalen University, Sweden
wasif.afzal@mdh.se

Daniel Sundmark
Mälardalen University, Sweden
daniel.sundmark@mdh.se

ABSTRACT

Background: Testing is one of the main methods for quality assurance in the development of embedded software, as well as in software engineering in general. Consequently, test results (and how they are reported and visualized) may substantially influence business decisions in software-intensive organizations. **Aims:** This case study examines the role of test results from automated nightly software testing and the visualizations for decision making they enable at an embedded systems company in Sweden. In particular, we want to identify the use of the visualizations for supporting decisions from three aspects: in daily work, at feature branch merge, and at release time. **Method:** We conducted an embedded case study with multiple units of analysis by conducting interviews, questionnaires, using archival data and participant observations. **Results:** Several visualizations and reports built on top of the test results database are utilized in supporting daily work, merging a feature branch to the master and at release time. Some important visualizations are: lists of failing test cases, easy access to log files, and heatmap trend plots. The industrial practitioners perceived the visualizations and reporting as valuable, however they also mentioned several areas of improvement such as better ways of visualizing test coverage in a functional area as well as better navigation between different views. **Conclusions:** We conclude that visualizations of test results are a vital decision making tool for a variety of roles and tasks in embedded software development, however the visualizations need to be continuously improved to keep their value for its stakeholders.

CCS CONCEPTS

• **Software and its engineering → Software creation and management**; **Software testing and debugging**; **Empirical software validation**;

KEYWORDS

Software Testing, Visualizations, Decision Making

ACM Reference Format:
Per Erik Strandberg, Wasif Afzal, and Daniel Sundmark. 2018. Decision Making and Visualizations Based on Test Results. In *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '18), October 11–12, 2018, Oulu, Finland*. ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3239235.3268921

ESEM '18, October 11–12, 2018, Oulu, Finland

ACM ISBN 978-1-4503-5823-1/18/10.
https://doi.org/10.1145/3239235.3268921

1 INTRODUCTION

Testing is a fundamental quality assurance activity in software development. The results of testing are typically evaluated in different ways to assess the quality of the software under test. Major unexpected outcomes of testing typically cause increased time and effort to isolate, identify and describe the unexpected outcomes.

Test results analysis and supporting visualizations have slowly gained importance, as the complexity of developing software has increased. Furthermore the stakeholders now need reliable data to base their critical decisions upon. Within the current trend of continuous development [20], the goal of continuous testing is to provide rapid feedback on the possible success of the latest build or release candidate. Test results are typically recorded in some form. A test results database (TRDB), with supporting analyses and visualizations, may serve as one important success factor in ensuring the provision of this rapid feedback. Some information regarding test execution can also be entered in test logs that can capture the timing of test execution and configuration used when testing [13]. It is important to distinguish TRDBs from bug/issue reporting and tracking databases such as BugZilla and Mantis BT. This separation is especially interesting since most of the research within software engineering has historically focused on data from bug/issue reporting and tracking databases [27], while leaving the potential of exploiting TRDB for decision making largely untapped.

The objective of this paper is to explore the use of test results analysis & visualizations in the context of complex embedded system development. We do this by conducting an industrial case study at Westermo Research and Development AB (Westermo), which designs and manufactures robust data communication products for mission-critical systems. We specifically address how Westermo utilizes the TRDB as an aid in decision making at several critical points in the development of its industrial network operating system, the Westermo Operating System (WeOS). These critical points are the daily work, feature branch merge and release time.

The TRDB at Westermo has helped different stakeholders to navigate through an increased complexity arising from a growing number of parallel code branches, an increasing number of test cases and an increasing number of test systems with different configurations. The role of the TRDB is further enhanced in time-pressure situations; e.g., near release times, several visualizations made possible by the TRDB increases confidence in the critical decision for whether or not to release a particular WeOS version.

2 BACKGROUND AND RELATED WORK

Recent international standards and the International Software Qualifications Board (ISTQB) both highlight the importance of communication for testers. Annex E of the ISO/IEC/IEEE 29119-1 standard [14] acknowledges that testers need to communicate with

various stakeholders in a timely manner and that this communication may be more formal or oral. ISTQB has released a syllabus relevant for test automation engineers [3] that includes a chapter on reporting and metrics where they recommend to visualize results, create and store logs from both the system under test and the test framework, and generate reports after test sessions.

Shahin et al. found that when continuous practices are adopted at an organization the frequency of integrations increase, and there is an exponential growth of information. They also found that lack of awareness and transparency is a challenge [20]. Regression test selection (covered in recent surveys [7, 12, 26]), is a common approach to make good use of the available test environment, given a time budget. Our previous work on system-level regression test selection describes Westermo's implementation [22, 24]. Just like what Shahin et al. suggest, we believe that regression test selection increases the burden of information overflow.

Metrics for the testing phases have been studied, e.g. in [1, 16]. Kan [16] argues that several basic testing metrics such as test progress curve, defect arrival density, critical problems before product ship, should be an integral part of all software testing. Kan is further of the view that testing metrics lack industrial validation: "*...few [testing metrics] are supported by sufficient experiences of industry implementation to demonstrate their usefulness*".

Within the field of software engineering, visualizations have been important for many decades. One survey on software visualization present methods for visualizing static aspects with the focus of source code lines, class metrics, relationships and architectural metrics [6]. An important publication using test data combines source code, code coverage, unit tests and test results in a sort of extended Integrated Development Environment (IDE) view [15]. A dashboard is a common approach for quality monitoring and control in the industry. These aim at presenting one or more key performance indicator (KPI) over time [10].

Recent studies on visualizing test results show that: (i) there is a need for summaries [17], (ii) information may be spread out in several different systems [4], (iii) one might need to consider transitions between different views of a test results visualization tool [18], and that (iv) visualizations may target managers, experts, testers and code maintainers, with the purpose of giving early warnings, suggestions for code improvements, planning support and support for making decisions [21]. Many of these studies also propose visualization methods. Some other examples of visualizations in the testing context are given in [8, 9] to identify similarities across test cases. In [18], the authors propose an interactive visualization technique for analysis of results of large-scale software regression testing. They propose multiple views for test results visualization, e.g., comprising of charts and tables.

One pattern we observed in test visualization papers is the lack of long term industrial validation. Related studies typically either do a proposal for a visualization, or a one-time evaluation. This paper is based on a test results database and visualizations that have been implemented over a duration of six years.

3 INDUSTRIAL CASE STUDY DESIGN

The objective of the case study is to explore the use of the TRDB and the visualizations it supports for the three critical decision points of: daily work, merging a code branch to the master branch, and at release time. The objective is refined into the following set of research questions:

- RQ1: How are visualizations used for making decisions in daily work, at merging a code branch to a master branch and at release time?
- RQ2: How do industrial practitioners perceive the value of visualizations, both in general and with respect to decision making in daily work, at merging a code branch to a master branch and at release time?

3.1 Case, Subject Selection & Industrial Context

Based on our stated case study objective and the RQs, the purpose is both exploratory and descriptive (finding out what is happening, describing the current situation, and generating improvement ideas). The case in our context is the test results visualization implementation supported by the TRDB at Westermo. We do not restrict ourselves with a particular project or a software release as the visualizations are available as a support across all projects and releases. However, the units of analysis in our case are the different roles we interviewed and sent questionnaire to. These roles are a project manager, a test manager, two developers and one test environment developer. Thus one can think of this case study as an embedded case study with multiple units of analysis [19].

Westermo provides robust and reliable data communication network devices (such as Ethernet switches, modems and routers) to clients ranging from transport, water and energy supplies to process industries, such as mining and petrochemical. Their products and services are for mission-critical systems operating in physically demanding environments where network outages can have dire consequences. On the software front, they build the industrial operating system WeOS that powers its products and helps in configuration and maintenance.

The test results are gathered from the nightly system-level regression testing of WeOS running on one or more devices under test (DUTs) in a real network setting. These test results are stored in a database for generating different visualizations. The following Sections provide more information on the development and testing process as well as the TRDB in use at Westermo.

3.1.1 Embedded Software Development and Testing Processes. At Westermo, the software development is an agile feature driven process based on Kanban. Features are developed in isolation from each other where each feature development constitutes a feature branch. Each feature branch goes through different phases of development. Once all phases are done, it becomes a candidate to be merged in the integration (or the master) branch (Figure 1). The system-level regression testing of WeOS is done nightly when no one is doing any software changes. This regression testing is an important part of Westermo's continuous integration framework. To provide an environment for both manual and automated test execution and reporting, Westermo has implemented and maintained a test framework over a period of several years. The test framework allows for test results recording in an automated fashion. To run the system-level nightly regression testing, a number of test systems are built as part of the test environment. These test systems are built as different network topologies, which in turn correspond to the

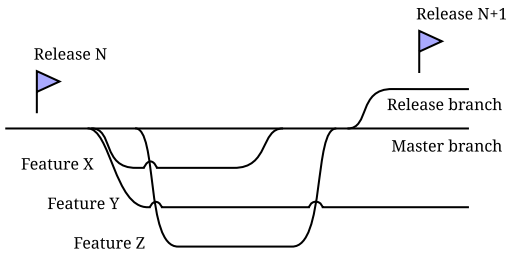


Figure 1: Feature driven development at Westermo. After Release N three features were started: X, Y and Z. For Release N+1 only X and Z were included (merged), and feature Y will be kept in its own feature branch until it is ready for release.

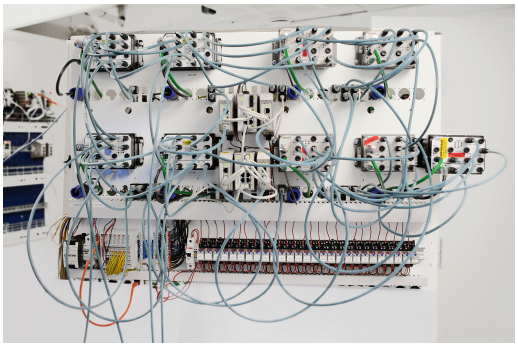


Figure 2: An example test system at Westermo.

different scenarios that these DUTs will be part of in a real network setting in an actual field operation. Figure 2 shows an example of one such test system having some WeOS devices, a PC server and a few other devices of other types.

The development of these dedicated physical test systems is one major aspect differentiating embedded software development from traditional software engineering.

The DUTs in the test system communicate via different protocols and through different ports using cables such as Ethernet and optical fibre. Each DUT runs one of several customized versions of WeOS. WeOS is composed of several software libraries that include several source code files.

The nightly testing uses the latest WeOS images; any changes to WeOS are pushed to the source code repository prior to the start of nightly testing during the day time. The test suites are built and prioritized using Westermo's internally developed test prioritization tool *SuiteBuilder* [22, 24]. The test cases in each test suite are mapped to different test system network topologies [23] and then executed one by one. The verdict of each test execution is reported to the TRDB.

3.1.2 Test Results Database. Westermo's test framework runs on average about 3500 to 4000 tests on more than a dozen test systems on many versions of WeOS every night. In Figure 3 this is illustrated in terms of number of test systems, test cases, and code branches in use, as well as number of outcomes, all broken down per quarter. When the TRDB was first implemented in 2012, the nightly

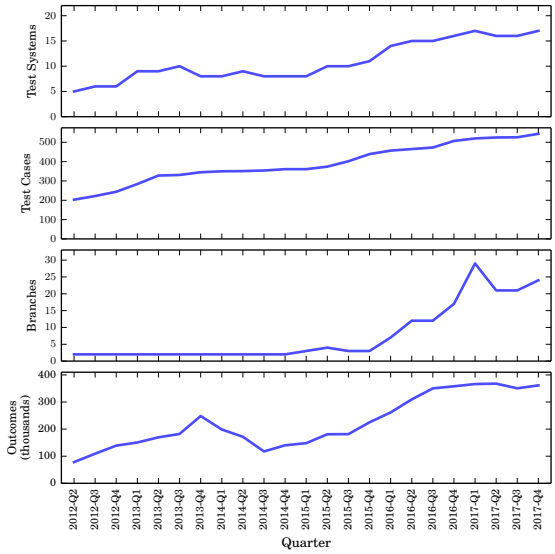


Figure 3: Test results data complexity: number of test systems, test cases, code branches, and test outcomes per quarter.

testing was conducted using 5 test systems, on 2 code branches with 203 test cases, or roughly 2.000 theoretical combinations. In Q4 of 2017, there were 17 test systems in use, 24 code branches, and 544 active test cases or roughly 220.000 theoretical combinations thereof. From Figure 3 one can make three important observations: (i) The number of test systems and test cases roughly grows linearly over time. (ii) The number of code branches was stable at two until 2016 when the feature driven development process was introduced, and since then the number of branches in use each quarter roughly grows linearly. (iii) The number of outcomes (verdicts from started test cases) grew linearly until end of 2013, and had a dip in 2014. At this time nightly testing did not finish in time, and test selection was introduced [22, 24]. Then the number of verdicts grew until 2017 with an increased number of test systems, and in 2017 the number of test systems plateaued.

The test results statistics are stored for each test case execution (outcome) in terms of passed, failed, invalid, unmappable, unloadable or skipped tests. 'Invalid' tests are those tests that failed because of an unhandled exception or a possible shortcoming in the test framework or the test environment. 'Unmappable' means that physical resources were not available to run some tests. 'Unloadable' tests are those that went missing for some reason or had syntax errors in the Python code. 'Skipped' tests were low-priority test cases towards the end of the suite that could have been executed if more time had been available.

The results generated are non-trivial to understand, owing to this multi-dimensional complexity seen already in 2012. There was a need for a multi-purpose relational data base such that one could query relevant information, depending upon one's role in the company, i.e., a developer's information needs are different than that of a test lead or a manager. Westermo's TRDB provided the foundation

to meet such stakeholder's needs. The current TRDB thus aims at: storing results from all nightly testing; storing enough information to recreate the state used on the DUTs and test servers when the test ran; allowing different views on the data, e.g., statistics on suite level, statistics per test, statistics per DUT-type, statistics per release; as well as generating condensed test reports.

There are a number of tables in the database that serve different purposes, such as storing information about the nightly sessions, tests and outcomes. This schema enables answering a number of fundamental questions about the nightly regression testing, such as: what was the outcome of running a particular test? and what were the states of the DUTs when a certain test case was executed?

3.1.3 Implementation of Tools. Different kinds of reports are available from the TRDB such as summary and detailed reports on nightly testing (e.g., finished tests/suites and duration per test/suite), test logs showing how a test has performed in the last 60 days. In addition to the reports, there are several plots generated (details on them are given in Section 4.1). The reports and the various visualizations based on TRDB are a result of three implemented tools: (i) *PaperBoy* is the initial tool that generates plain text summaries and basic trend plots. (ii) *TestAdm* is the most used tool, it introduced clickability and the possibility to see and filter log files in a browser. (iii) *PaperBot* is a set of command-line based tools that were first implemented to be an IRC chatbot, but that now only supports the command line. The code base in *PaperBoy* also generates heatmaps of test results. *PaperBoy* and *PaperBot* are both implemented in Python. *TestAdm* is implemented mostly in JavaScript on top of AngularJS, with some components in Python. All three tools communicate with the TRBD using hand-written SQL-queries embedded in the source code.

3.2 Data Collection

In order to approach the research questions from different angles and to ensure completeness of data collection, we used methodological triangulation by using multiple data collection techniques: interviews, questionnaires (as follow-up to interviews), archival data and participant observations. An overview is shown in Figure 4 and details explained below.

Interviews: We conducted a total of 5 semi-structured interviews with one project manager, one test manager, two developers and one test environment developer. We selected these persons based on convenience sampling as the industrial co-author of this paper, in consultation with the test manager, knew the most knowledgeable persons to interview in the company. We planned a series of questions before the interview but did not necessarily ask them in the exact order every time, but made sure that every question was asked. We started each interview with background questions about the interviewee, before asking specific questions related to our research objective. Two authors participated in each of the 5 interviews and also took personal notes. The interviewees had access to the test results visualization web page while they answered questions, and we encouraged them to use it during the interview, such that references to particular pages and figures can be easily made. The interview questions are given in Appendix A.

Questionnaires: We followed the interviews with a web-based questionnaire to the 5 interviewees. The questionnaire was kept small and included questions on their degree of satisfaction with existing visualization support for making decisions in their daily work, at branch merge and at release time. Specific questions were asked about ease of understandability of visualizations, ease of learning the meaning, time taken to visualize and ease of navigation. Each question had an optional space for including any free-text comments. The used questionnaire is given in Appendix B.

The questionnaires were given to the same persons we interviewed to serve two purposes: (i) to complete and complement the information given in interviews (ii) to be able to elicit usefulness of test visualizations, their ease of navigation, learnability, understandability and response times.

Archival Data: We had access to internal documentation as well as source code repositories from past and present projects. Documents such as formal process definitions and more informal internal wiki pages were also available. The archival information was mainly used to get familiar with Westermo's development process, the meaning of various internal classifications of test results as well as getting an overview of the database schema, associated script documentation and generated visualizations. This was essential to confirm, understand and add to the data collected from interviews and questionnaires and also helped us answer *RQ1*.

Participant Observations. Twice a week at Westermo a team of people with different roles gather around a large screen in a conference room for a sync meeting where test results are investigated by using the TRDB and the visualizations. The results are discussed and tasks of various kind, typically fault finding, are distributed. This is an important ritual for sharing knowledge like "Test system X is new and the devices in the system are early prototypes." We participated in two sync meetings and made high-level notes on how the visualizations were used and the shortcomings encountered.

3.3 Ethical Considerations

Several steps were taken to ensure that our data collection remains within known ethical guidelines. Co-authors from academia had signed an NDA prior to getting access to archival data. We informed the participants with the purpose of the interviews and questionnaires, both through email and at the start of each interview/questionnaire. Before recording an interview's audio, permission was taken from the interviewee. It was made clear to the interviewees that the audio files would not be uploaded on any server and only stored on local machines of the researchers. It was also conveyed to the interviewees that the audio files and transcripts would only be used to answer our research objectives in an anonymous and confidential manner.

3.4 Data Analysis

The interview data was transcribed and analyzed with thematic analysis [5]. We also familiarized ourselves with the interviews by listening to the audio files and reading the transcripts repeatedly. During the five interviews, we recorded 3.95 hours of audio that we transcribed into 19 dense A4 pages. During the transcription, names of products, people and tools were anonymized in case the

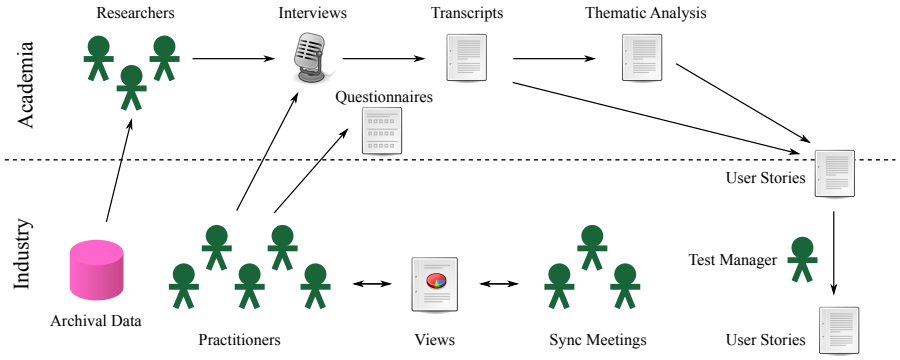


Figure 4: An overview of methodological triangulation used for the data collection.

transcript would be forgotten in a printer, etc. We identified 17 themes, and categorized interview snippets related to a theme as positive, neutral, or negative. For example, in one of the interviews the interviewee said "*[Looking at log files] is very valuable, the part where you can see the <device communication> log. I'd like to have the functionality to see the kernel log, and those kinds of things. But there is a penalty to collect these.*" This was clearly a discussion on the theme *logs*, that was both positive (it is good to have easy access to logs) and negative (there could be more logs stored).

The participant observations at sync meetings were summarized on a higher level than the interview transcripts, and were categorized in the same way as interview snippets. For example, during one sync meeting we noticed that the participants did an in-browser zoom of the web page and reacted when the font appeared to be broken. This was noted as "poor font zoomability" and categorized as negative under the theme of *UI*.

The data from the questionnaires were on a Likert scale, which was used to show respondents perception of the usefulness, ease of navigation, learnability, understandability and time consumption of visualizations. The archival data available through, e.g., the documentation on internal wiki pages, was used to explain the existing visualization support and was primarily used by the authors to both get familiarized with context and to understand the meaning of interview/questionnaire data.

3.5 Validity Analysis

The first author of this study has been part of the test team at Westermo for many years. Furthermore, the second author has regularly been located at the company for more than a year, and the final author regularly visits it. There has thus been a *prolonged involvement*, and the researchers of this study understand the vocabulary and viewpoints of the interviewees. However, all research has shortcomings, here we discuss some threats to validity based on the viewpoints proposed by [19].

We primarily investigated the constructs *decision making*, *role* and *visualization* and the relationship between these. Threats to the construct validity include the limits of scope: decision making was limited to three key points in the development process; we limited ourselves to interview five individuals with the four roles of developer, tester, project manager and test manager; and we limited the visualizations to already existing visualizations. However, we opted for interviewing key individuals belonging to different groups, and used the knowledge of the industrial co-author and the test manager as a criterion for interviewee selection.

The study was conducted at Westermo, an embedded systems company in Sweden, doing agile software development, with a strong focus on test automation, using an internally developed test results database. Results on the available visualizations at Westermo are of course not generalizable, but we provide many generalizable observations on how an individual having a role can use a visualization of test results in order to enhance decision making.

In order to improve validity of the results we collected data from several sources with different techniques (method triangulation) and none of the interviews were conducted by only one researcher (observer triangulation). Further, by describing the case and providing both the interview questions and the questionnaire, we have simplified the possibility for a replication study.

4 RESULTS

This section presents the results of the study as well as answers the stated research questions.

4.1 RQ1: Visualizations for Decision Making

This section presents how the TRDB and the visualizations are helping Westermo in three dimensions of daily work, at branch merge time and at release time.

4.1.1 Support in Daily Work. For the developers and the testing team at Westermo, the visualizations built on top of the TRDB are a tool to identify problematic areas and to identify the root causes of failed test executions. The visualizations help achieve this in a stepwise way, moving from giving a general picture for all branches to log files for a particular test outcome (Figure 5).

The very first visualization that a user can see is an overview of the WeOS branches that ran nightly testing, illustrated in Figure 6. These test results are presented in a summary/overview form using a circular progress graphic. Different colors in the circular progress graphic depict progress of different test results, e.g., green shows passing of tests, red shows failure while orange indicates that some tests remained invalid, unmappable or unloadable. A small color-filled inner circle shows the overall state of nightly

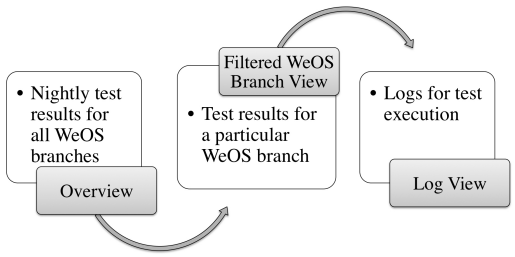


Figure 5: Hierarchical visualizations helping daily decision making based on the TRDB at Westermo.

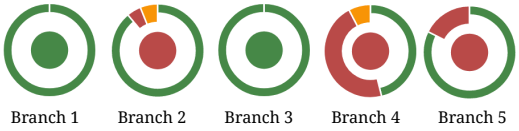


Figure 6: Overview: Each circular graphic represents a WeOS development branch.

testing as having completely passed or not, depending upon the outer, circular progress graphic.

The user has the possibility to go deeper into the test results for a particular development branch through an available menu where the different branch names are populated. Once the user clicks on the development branch of interest, the Filtered WeOS Branch View shows up (Figure 7). The default filter shows only failing tests for the last night. Filtered WeOS Branch View presents a more detailed visualization of test results for a particular branch on different test systems. Here different filters are implemented to help the development and the test teams. Under the current view, a developer or a tester can filter, for example, on number of days and type of test execution outcome (passing, failing, invalid, etc.). The ability to focus on a particular WeOS branch, as shown in Figure 7, is important for the development team. This allows both the developers and the testers to narrow down their work specific to a certain branch, out of the many parallel WeOS branches.

If a particular test outcome is shown to fail on a particular test system, the user is able to click on it to get further details. Clicking takes the user to the 'Log View' where one can read the test log messages to reach the root cause of a certain test execution result. One example of the log view is shown in Figure 8.

In addition to the web-based test results exploration, a summary report of each test suite of the nightly regression testing is compiled into a PDF and stored in a shared location at Westermo each working day. This report is occasionally used internally, and sometimes used in communication with customers.

4.1.2 Support at Branch Merging. Merging a feature branch in the master branch is a critical decision to make. In order to be ready for the merge, all automated nightly tests on the feature branch should be passing. Moreover, the feature branch is considered stable when a number of nightly tests are found to be passing consecutively for a number of days. One of the most important

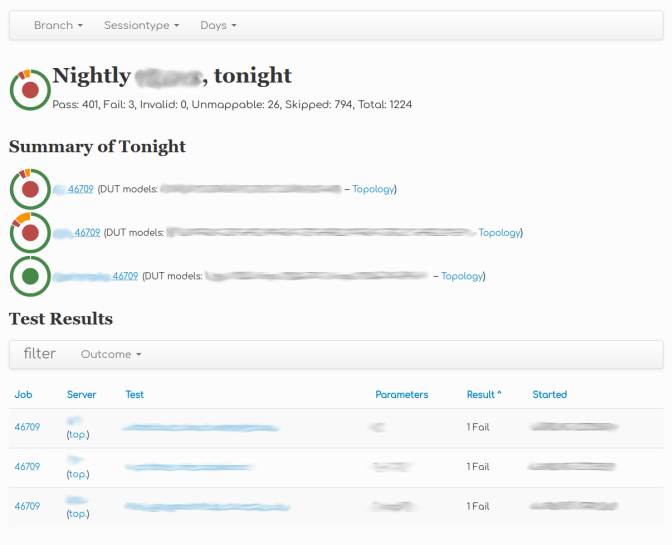


Figure 7: Screenshot from a filtered WeOS Branch View: Test results for different test systems with test statistics. (Sensitive information has been blurred for confidentiality.)

```
03:25:34,985 Test Environment: Loading topology
03:25:36,387 Test Environment: Reset of devices OK.
03:25:38,110 Test Environment: Mapping complete.
03:25:38,322 Test Case: Speed test of ports 1, 4 and 8.
03:25:40,532 Test Case: Bad speed at port 4. Fail!
03:25:40,672 Test Case: Cleaning up.
03:25:42,137 Database: Writing results.
```

Figure 8: Log View: An example test log.

visualizations for assisting decision making in merging a feature branch is the trend plot of a certain feature area on all the test systems. In Figure 9, two example trend plots are presented.

Another visualization supported by the TRDB that provides important information regarding the health of a feature branch is the heatmap of a particular test case over various test systems in the past few weeks. Three examples of such heatmaps are given in Figure 10: the first one shows the introduction of a new test case, then one that shows a test case that fails on multiple test systems, and last a test case with intermittent invalid results – a "flaky" test.

4.1.3 Support at Release Time. A set of features, when stable and merged in the master branch, allows for a WeOS release to be made. From a test perspective, all the features going into the release should be performing correctly and have been integrated successfully. Furthermore, testing at the release phase is done according to a risk analysis. For all the features included in the release scope, the risks identified for all integrated features from previous test phases need to be evaluated from a release perspective. The TRDB and the supporting visualizations help achieve this evaluation.

There is a set of command line scripts that a test manager runs on the TRDB to query long-term statistics. Two of the most important queries are the 'trend' and 'matrix' scripts, which both present results from all or a few features, and on all or selected test systems,

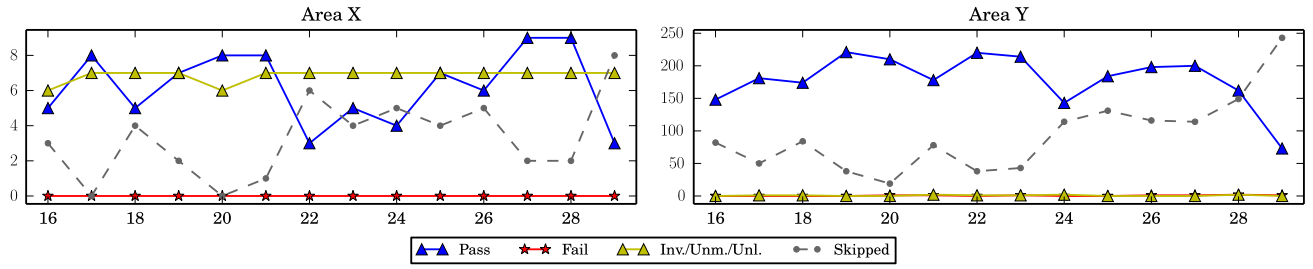
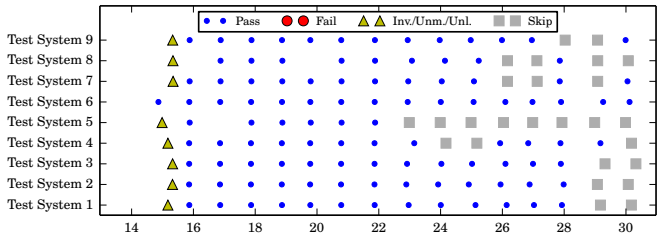
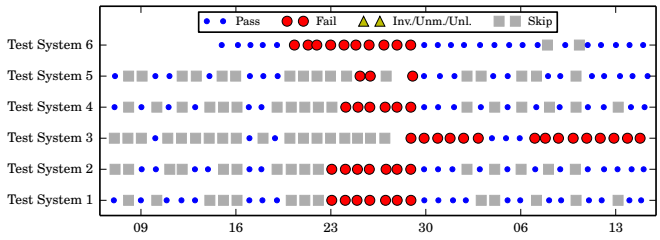


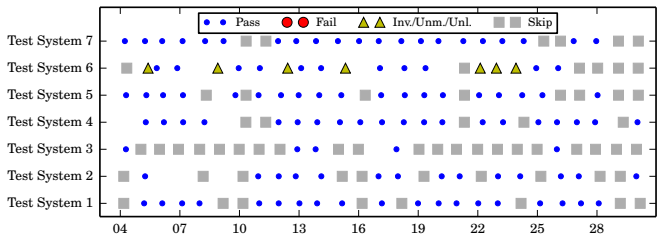
Figure 9: Trend plots for two different functional areas helping to decide its merging or not with the master branch. Area X has some invalid results, whereas Area Y only has passing (and skipped) test cases.



(a) The introduction of a new test case.



(b) A test case infrequently being tested in nightly (many gray squares), that then fails on all test systems for several nights. After some modifications it is corrected on all test systems except Test System 3 where it continues to fail.



(c) A test case with intermittent problems on Test System 6.

Figure 10: Heatmaps illustrating three failing patterns. All figures have test system on the Y axis and time on the X axis.

on the release branch in the last few days. Examples with output is shown in Figures 11 and 12. The test manager can look at the results in different dimensions, such as how did a certain feature perform on testing for last few days. Similarly, the state of test systems can be assessed, which is important if new hardware was introduced.

```
$ trend -b branch -a area -d 10

    Date      Pass   Fail   Inv.   Unm.   Unl.   Skip
 201X-0Y-01     84      0      0      0      0     77
 201X-0Y-02    125      0      0      0      0     36
 201X-0Y-03    161      0      0      0      0      0
 201X-0Y-04    137      0      0      0      0     24
 201X-0Y-05    155      0      0      0      0      6
 201X-0Y-06    143      9      1      0      0      8
 201X-0Y-07    106      0      0      0      0     55
 201X-0Y-08     63      0      0      0      0     98
 201X-0Y-09     59      0      0      0      0    102
 201X-0Y-10     29      0      0      0      0    132
```

Figure 11: Output of the trend script. The optional argument -a has filtered the results given a specific functional area.

```
$ matrix -b branch -d 7 -s system1 -s system2 -s system3
-a area1 -a area2 -a area3

 Area\Sys.    system1      system2    system3       total
   area1    80 of 80   130 of 130   23 of 23   233 of 233
   area2     0 of 0      7 of 7      0 of 0      7 of 7
   area3    18 of 18    42 of 42    17 of 18    77 of 78
   total    98 of 98   179 of 179   40 of 41   317 of 318
```

Figure 12: Output of the matrix script. The argument -b selects the branch, -s selects test systems, -a selects functional areas, and -d selects number of days to investigate.

The matrix statistics show the number of tests passing for a certain combination. "17 of 18" for area3 on system3 means that one test failed in this branch, during these days, on this test system for this feature.

If there is a significant decrease from 100% in a row, it could mean that this feature is not mature. But if there is a significant decrease from 100% in a column, it could mean that this test system is either faulty or not configured properly. If there are failures all over the place in the matrix, then this is an indication that this is not a good release candidate at the moment. If there is a "0 of 0", it could mean that this combination should either be tested more, or that this system does not support testing of this feature.

Other scripts and visualizations on the TRDB are available to the test manager at the release time, such as what failed in the last few nights, how did an area of interest perform in nightly testing in past few days, the already mentioned heatmap plot, among others.

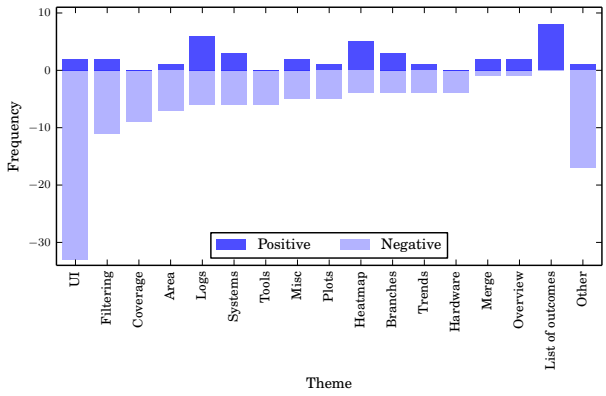


Figure 13: Themes frequently discussed in interviews.

4.2 RQ2: Perceived Value of Visualizations

All the roles we interviewed utilized the visualizations with varying degree of intensity. There was an unanimous agreement on the utility and usefulness of visualizations, although each one of them also expressed several areas of improvement. In this section, we begin with a discussion of themes frequently discussed in interviews (8 out of 17) and categorize them as being positively or negatively perceived by the interviewees. Later we analyze the value of the visualizations as told by the interviewees at the three decision points of daily work, feature branch merge and release time. Lastly, we summarize the data from the questionnaires as a frequency bar chart to show respondents perception of the usefulness of visualization as a support for decision making, its easy of navigation, learnability, understandability and timing.

4.2.1 Positive & Negative Themes. In Figure 13, we summarize the aggregated themes identified in the interview data with their positive and negative ranking (neutral removed). Below we summarize the most important findings of the most important themes.

User Interface: The most discussed theme was on the User Interface, UI. During the interviews the interviewees had problems navigating and also expressed frustrations on the inability to use the browser back button and poor clickability (in terms of understanding what elements are links and not being able to click interesting pieces of plots), poor positioning of filter elements (making it easier to change the URL than clicking a button), and poor fonts when zooming (for example when looking at the tool on a big screen during a sync meeting).

Filters: Filtering is important, and the default filter (hiding passing tests) was perceived as good. There were however a strong desire for being able to filter in ways not supported, such as removing results from new (and experimental/unstable) test systems, or seeing only results from a certain hardware product type.

Coverage: There are several important challenges in understanding coverage: (i) test cases are selected dynamically with SuiteBuilder so not all test cases are executed every night, (ii) some test cases are not relevant for all test systems nor for all code branches, (iii) focus is mostly given to failing tests, and (iv) test cases that trigger known issues are not run in the regular nightly suite, instead they go in a separate suite for the known issues. Not knowing if a test is executed, filtered away or not in scope for a test system or branch, was an uncertainty several interviewees mentioned.

Area: All functional areas in WeOS has at least one developer dedicated as responsible for it. Interviewees requested better support by the tool for these individuals. In particular: a good way to see coverage of the test cases in the area.

Logs: The tool can show log files that are timestamped so that sequences and other details can be explored. For the interviewees the ability to investigate log files was one of the most important features. However, the log files have different formats, and only one log file is easily seen at a time. Some log files could be filtered with respect to severity of the logged message, but there is no possibility to filter based on time, such as only showing what happened in a time range of five seconds before an error occurred.

Systems: Very few details on the test systems were shown in the tool at the time of the interviews. There is a desire to understand what hardware there is in a test system, how cables are connected, and if the system has been modified recently. During this study the tool was updated to also include lists of models of the DUTs of the test systems, and also a visualization of the overall network topology of each test system.

Heatmaps: One of the themes with the most positive remarks was heatmaps, illustrated in Figures 10. These plots also received critique for being messy and one interviewee asked "*What do the gray boxes mean?*".

List of Outcomes: The only theme that received no negative opinions was having a list of outcomes with links to the log view of each test execution, failing or not. This is one of the most, if not the most, important feature of the tool and one interviewee said "*It helps us go through the results*".

4.2.2 Value in Daily Work, at Branch Merge and at Release Time. In daily work in the morning, one WeOS developer said that visualizing the results of nightly testing is the first thing he does, to get an estimate of the quality of the feature branch under development. On a daily basis, this WeOS developer was of the view that the ability to see the log of a failed test execution is valuable. The test environment developer, on the other hand, mentioned that the first thing he does in the morning is to assess the health of various test systems after nightly testing and to see if there are some test results missing from one or more test systems. He does this for a particular branch of interest. Failing tests are also of interest where he mentioned finding out if a test is failing on several test systems to indicate a possible problem in the test script itself. The second WeOS developer we interviewed said that he starts the daily work by checking the nightly testing results on a particular feature branch of interest. If some tests are found to be failing on the branch, a more detailed investigation starts, otherwise the passing test results are not looked into further. The detailed investigation concerns the areas he is responsible for. Some failing tests are known from before, so they are ignored while new failures are looked into before the daily morning stand-up meeting.

When it is time to decide on merging a feature branch to the master branch, the WeOS developer said that the test results visualizations are mostly valuable at this stage as a certain degree of

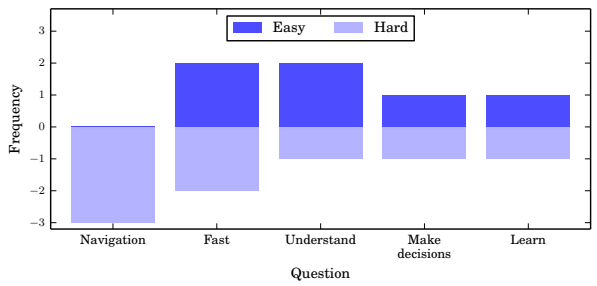


Figure 14: Answer frequencies to questions.

stability in the feature branch (shown by passing test results) is expected before its merge with the master branch. The second WeOS developer mentioned that close to the merge decision for a feature branch, a risk workshop is done where test results on several areas are seen and analyzed. The project manager emphasized that test results is a primary indicator of stability of a feature branch, so at merge time, he first checks if the test results on a certain test system are failing. Then sorting is done at the test case level to see if a particular test is failing on one or more test systems. Then the different areas are selected to analyze test results within them.

Close to release time, the test manager hinted at the criticality of viewing test results. She mentioned searching and evaluating test results several nights back to see trends. Test cases that fail constantly or intermittently receive troubleshooting to expose the cause of the failures. Test results of important functional areas are also looked at carefully to evaluate risks associated with the release. The project manager mentioned that for the release candidate, all the test results are reviewed and it is made sure that different required test systems have been running the test cases successfully. Functional areas are made sure to have undergone successful testing.

4.2.3 Answers in Questionnaires. All five interviewees answered all five questions in the questionnaire. None of the interviewees used the "Very hard" or "Very easy" option in any question. In Figure 14, we have plotted the "Easy" and "Hard" answer frequencies. Three aspects had symmetric answer frequencies (as many "Easy" as "Hard"): *Fast*, *Make decisions* and *Learn*. *Navigation* had no positive answer and three negative, whereas *Understand* had two positive and one negative answer. The analysis from the questionnaires helped reinforce some of our findings from the interviews, e.g., the findings from the theme of *User Interface* from the interviews and *Navigation* aspect from questionnaires show that users need improvements in this aspect.

5 DISCUSSION

In his keynote at a practitioner-oriented conference, James Bach highlighted that stakeholders do not know what to visualize: "What I had to do, is use my skill as a tester, and my interest in visual design, and complexity, and displays, and statistics, to try to come up with something, that they, when they saw it – it would be like someone seeing an iphone for the first time …" [2]. This, anecdotally, suggests that visualizations are not prioritized and that individuals skilled in other domains are the ones that end up being the ones preparing visualizations.

The results in this paper have shown how visualizations and reporting based on the test results database are utilized in a complex development environment arising from a number of code branches, different hardware and different network topologies. This study captures three instances where a test results database and the supporting visualizations are being successfully exploited in a real industrial context: during daily work, before deciding to merge a feature branch to the master branch and at release time. During daily work, the results of the nightly regression testing can be analyzed through different views, moving from an overview of testing results on each development branch to the specific view of execution log for a test case. Trend plots and heatmaps for a feature branch show evidence in support of or otherwise regarding the decision to merge it to the master branch. At release time, test result trends, and a consolidated matrix view of the test execution results on various test systems across different features for a number of days helps the test manager in the decision to release or not.

In a recent literature review on recommendation systems for software engineering [11], the authors highlighted the gap that exists on the lack of decision support systems for software testing. Similarly, based on our limited search to find relevant papers on visualization and decision support for software testing, many papers were found to be still at a stage of proposing and/or developing a theory. This is a little surprising since visualization and decision support techniques have been around for a while. In fact, there is more progress in using such techniques for program and source code comprehension (see e.g., [25]); it is only timely that the testing phase catches up.

The development team, being the user of visualizations and test results, need to be continuously asked about their needs and value-adding features. One interesting outcome of this study is eliciting such needs from the team. Thus as a result of this study, Westermo has generated a list of user stories to implement for the improvement of decision-making and visualization support based on test results. The user stories have been generated based on the summary of transcript snippets. For example:

- Transcript: "*I don't change the filters to start with [...]* "
- Summary: "*Filters*: defaults are good to start with"
- User Story: "In my daily work; I want sensible default filters in all views; so I won't have to start by re-filtering."

We generated both functional user stories (as the one mentioned above) and also non-functional user stories like: "In the future, when we have much more testing, the tool must still be able to work without scalability problems." A total of 142 user stories were generated and also prioritized by the test manager at Westermo for determination of relevance. At the end, it was clear that the visualizations and test results analysis has to be a continuous improvement activity.

6 CONCLUSION

The test results and corresponding visualizations have the potential to help decision making across different stages in embedded system

development. The existing research work on exploiting test results and visualizations lack a long-term industrial validation and sharing of real-world experience. We have conducted an industrial case study in this paper on the topic of test results and their visualizations where Westermo have utilized them over a period of six years. In particular, we have shown how test results and visualizations are utilized during daily work, before deciding to merge a feature branch to the master branch and at release time. We have also shown how such analyses is perceived by industrial professionals having different roles. Based on our analysis, the overall perceived advantages of visualizations and reporting were confirmed by the industrial participants. Furthermore, several requirements to further improve the visualization experience were elicited. Finally, our results show how a variety of visualizations as well as reporting mechanisms based on the TRDB can serve as decision-support tools, especially in a complex development environment having a number of code branches, different hardware and network topologies.

ACKNOWLEDGMENTS

This work was supported by Westermo Research and Development AB, the Swedish Knowledge Foundation (grants 20130085, 20130258, 20150277 and 20160139) and the Swedish Research Council (grant 621-2014-4925). In particular we would like to thank interviewees and visualization co-developers at Westermo.

REFERENCES

[1] W. Afzal and R. Torkar. 2008. Incorporating Metrics in an Organizational Test Strategy. In *IEEE International Conference on Software Testing Verification and Validation Workshop*.
[2] James Bach. 2018. Beauty or Bugs: Using the Blink Oracle in Testing. https://www.youtube.com/watch?v=5W_VLzNhT-s Keynote at the Beauty in Code Conference, Malmö, Sweden 2018.
[3] Bryan Bakker, Graham Bath, Armin Born, Mark Fewster, Jani Haukinen, Judy McKay, Andrew Pollner, Raluca Popescu, and Ina Schieferdecker. 2016. *Certified Tester Advanced Level Syllabus Test Automation Engineer.* Technical Report. International Software Testing Qualifications Board (ISTQB).
[4] Martin Brandtner, Emanuel Giger, and Harald Gall. 2014. Supporting continuous integration by mashing-up software quality information. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on.* IEEE, 184–193.
[5] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101.
[6] Pierre Caserta and Olivier Zendra. 2011. Visualization of the static aspects of software: A survey. *IEEE tran. on visual. and comp. graphics* 17, 7 (2011), 913–933.
[7] Cagatay Catal and Deepti Mishra. 2013. Test case prioritization: a systematic mapping study. *Software Quality Journal* 21, 3 (2013), 445–478.
[8] Emelie Engström and Per Runeson. 2013. Test Overlay in an Emerging Software Product Line - An Industrial Case Study. *Inf. Softw. Technol.* 55, 3 (2013), 581–594.
[9] Daniel Flemström, Wasif Afzal, and Daniel Sundmark. 2016. Exploring Test Overlap in System Integration: An Industrial Case Study. In *42nd Euromicro Conference series on Software Engineering and Advanced Applications*.
[10] Maria-Elena Froese and Melanie Tory. 2016. Lessons Learned from Designing Visualization Dashboards. *IEEE comp. graphics and apps.* 36, 2 (2016), 83–89.
[11] Marko Gasparic and Andrea Janes. 2016. What recommendation systems for software engineering recommend: A systematic literature review. *Journal of Systems and Software* 113 (2016), 101 – 113.
[12] Dan Hao, Lu Zhang, and Hong Mei. 2016. Test-case prioritization: achievements and challenges. *Frontiers of Computer Science* 10, 5 (2016), 769–777.
[13] IEEE Computer Society. 2014. Guide to the Software Engineering Body of Knowledge (SWEBOK) V3.0.
[14] ISO/IEC/IEEE. 2013. *Software and systems engineering – Software testing – Part 1: Concepts and definitions.* ISO/IEC/IEEE Standard 29119-1:2013. International Organization for Standardization, International Electrotechnical Commission, Institute of Electrical and Electronics Engineers.
[15] James A Jones, Mary Jean Harrold, and John Stasko. 2002. Visualization of test information to assist fault localization. In *Proceedings of the 24th international conference on Software engineering*. ACM, 467–477.
[16] Stephen H. Kan. 2002. *Metrics and Models in Software Quality Engineering* (2nd ed.). Addison-Wesley Longman Publishing Co., Inc.
[17] Agneta Nilsson, Jan Bosch, and Christian Berger. 2014. Visualizing testing activities to support continuous integration: A multiple case study. In *International Conference on Agile Software Development*. Springer, 171–186.
[18] Rudolfs Opmanis, Paulis Kikusts, and Martins Opmanis. 2016. Visualization of Large-Scale Application Testing Results. *Baltic J. of Mod. Comp.* 4, 1 (2016), 34.
[19] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical SE* 14, 2 (2009), 131–164.
[20] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. 2017. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access* 5 (2017), 3909–3943.
[21] Per Erik Strandberg. 2017. *Software Test Data Visualization with Heatmaps.* Technical Report.
[22] Per Erik Strandberg, Wasif Afzal, Thomas Ostrand, Elaine Weyuker, and Daniel Sundmark. 2017. Automated System Level Regression Test Prioritization in a Nutshell. *IEEE Software 2017* 34, 1 (April 2017), 1–10.
[23] Per Erik Strandberg, Thomas J Ostrand, Elaine J Weyuker, Daniel Sundmark, and Wasif Afzal. 2018. Automated Test Mapping and Coverage for Network Topologies. In *Software Testing and Analysis (ISSTA), 2018 ACM SIGSOFT 27th International Symposium on*. ACM.
[24] Per Erik Strandberg, Daniel Sundmark, Wasif Afzal, Thomas J Ostrand, and Elaine J Weyuker. 2016. Experience Report: Automated System Level Regression Test Prioritization Using Multiple Factors. In *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on*. IEEE, 12–23.
[25] David A. Umphress, T. Dean Hendrix, James H. Cross II, and Saeed Maghsoodloo. 2006. Software visualizations for improving and measuring the comprehensibility of source code. *Science of Computer Programming* 60, 2 (2006), 121 – 133.
[26] Shin Yoo and Mark Harman. 2012. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Ver. & Rel.* 22, 2 (2012), 67–120.
[27] Tao Zhang, He Jiang, Xiapu Luo, and Alvin T.S. Chan. 2016. A Literature Review of Research in Bug Resolution: Tasks, Challenges and Future Directions. *Comput. J.* 59, 5 (2016), 741–773.

A INTERVIEW QUESTIONS

- Questions about the interviewee:
  - How long have you worked with IT?
  - What is your role at the company?
- General questions about the WeOS Release Process/day-to-day work/merging a feature branch:
  - How are decisions (based on nightly testing) made when releasing a new WeOS version? Or How do you decide on what to do in the morning? Or How do you decide on merging a feature branch to a master branch?
  - Do you have a internal document that describes it?
- Specific question about the visualization aspects that support three critical decision-making points:
  - How do you use the visualizations to help you near the WeOS release time/in day-day work/at merging a feature branch to the master branch?

B QUESTIONNAIRE

All the questions below had a Likert-scale answering options (strongly agree, disagree, neither agree or disagree, agree, strongly agree) and free text space for optional comments.

- Do you think the existing visualization support is enough for you to make a sound decision for WeOS release/in day-day work/at merging a feature branch to the master branch?
- How easy is to understand the different visualizations that helps you near the WeOS release time/in day-day work/at merging a feature branch to the master branch?
- How easy is to learn the meaning of different visualizations that helps you near the WeOS release time/in day-day work/at merging a feature branch to the master branch?
- How time consuming is it for you to use the visualizations?
- How easy is it to navigate between the different views/pages of the web-based visualization support?