Test Effectiveness, Localization, and Prioritized Generation in Complex Test Environments

TODO: I have other sections like intro, but let's just focus on the milestones for now.

Background: Overview of Research Approach and Data

1 Research Milestones

This research proposal builds upon an NSERC Engage and an ongoing MITACs. For each milestone, we describe the work that we have done over the last six months and the direction we plan for this proposal. The proposal consists of TODO: X milestones. In Milestone 1, we will evaluate test effectiveness and build a test prioritization model. In Milestone 2, we will use the test failure distribution to find anomalies and differentiate between product and environmental faults. In Milestone 3, we will improve fault localization by mining test logs. In Milestone 4, we integrate our results and use them to quantify the change risk associated with test failures and relate them to customer found defects to identify which tests should be run for a change, which tests should be run after a failure, and which tests should be written to stop fault slipthrough to customers. We integrate our work into a framework that provides actionable, timely results to Ericsson developers, we call this tool TestGuru.

1.1 Test Effectiveness, Prioritization, and Groups

The complex test infrastructure and massive number of tests that run each night makes it difficult for Ericsson testers and managers to determine how effectively they are testing their software. We determined test effectiveness and initial test prioritization schemes. We plan novel work on prioritization in the context were not all tests are independent.

We began our NSERC Engage by extracting, cleaning, and linking test runs with problem reports and the version control system. Ericsson managers wanted to understand two basic questions: which software units (areas of the system) have the largest number of failing tests, and which tests find the largest number of faults. This initial work took a researcher associate, Zhu, four months. The industrial outcome of this work has been significant. We found that 25% of faults find 80% of the bugs that were identified by testers as true defects. This helped test managers in their ongoing test re-writing and re-organization efforts. Further, the development team received a list of their most common software units which lead to discussion of the cause of the failures allowing them to mark code and tests for re-work. TODO: More from Gary and Chris.

From a research perspective, our results replicate findings in a new context. First, as Mockus *et al.* [] found, having high test coverage comes at the cost of high redundancy. This redundancy results in high maintenance overhead and does not necessarily mean that fewer customer found defects will be caught by tests. Proposed work on this finding is discussed in TODO: Section. Second, our findings supports research into that prioritize tests based on past runs [**?**, **?**]. In this proposal, we constrain possible prioritizations/reorderings into dependent groups of tests to reduce the false positives introduced by inappropriate reorderings.

Existing work on test prioritization usually assumes that tests are independent and can be run in any order. In test environments that involve complex hardware and software, the order of test cannot be assumed to be independent. For example, an effort by Ericsson TODO: Sweden to reorder and re-write test to be independent was done at TODO: great cost etc, was it successful, the main challenge was to ....

Dependent Groups of Tests

We propose a novel research approach that groups sets of dependent tests into atomic units that can be reordered only as as a group. First, we plan to run simulations on past test runs to show that the new ordering would have found faults earlier and with less test runs. The simulation is evaluated based on the number of slipthroughs, *i.e.* tests that would have been run but were not that found faults, vs how much earlier a fault is found and how many fewer tests need to be run to find the fault. In the simulation, no new tests are run, so we assume that tests are independent and can be ordered arbitrarily. If the simulations show that the benefit of re-ordering is low, then the product team need will not need expend any further effort and can keep the existing order.

Second, to remove the test independence assumption, we must run actual tests with a new order. At Ericsson tests are expensive and we cannot re-run the entire set of tests across old products. Since we must conduct our experiment of test ordering in a live environment, there are two possibilities when a failure occurs. First, the test order is not valid. Second, a fault has been found in the system. To deal with these two conditions, we design our experiment as follows:

1. prioritize tests on the basis of their past test failure distribution []. We also plan to use other historical information, see TODO: Section.

2. if a group of tests fail, re-run in the normal order. If there are no fails on re-run, then mark the tests as dependent. If there are fails on re-run, send the failures to Ericsson testers for investigation because it is likely true failure. Note, failing tests are automatically re-run at Ericsson, so the additional cost will be minimal.

3. if the tests pass, mark them as independent.

Initially, there will be large number of failures that will be the result of inappropriate orderings. However, as our dataset grows, we will be able to determine the likelihood that a failure is related to a defect vs a failure related to an inappropriate ordering. Tests that fail together more often than by chance will be grouped into atomic units. These groups of tests will be reordered as a single grouping.

The final stage of this research will be to simulate the added benefit of breaking a test group. We will simulate the time savings and compare it with the test dependencies to provide an ordered cost benefit analysis. This simulation will allow Ericsson managers to determine which test clusters to re-structure first.

1.2 Anomalies: Environment vs Product

Fault localization is a wide research area including debugging techniques and statistical bug models, so in the NSERC Engage work, we focused on the narrow question – Did that test fail because of a problem in the development environment or product? This problem is unusually difficult at Ericsson because the environment is complex. As a result, there are TODO: eight Ericsson Ottawa developers who examine the failures that occur in the system after the nightly test run. Our goal in this milestone to help the team differentiate between environment and product faults and to prioritize the failing tests for investigation.

The expected novel contributions in this milestone are twofold. First, most prioritization techniques determine the order of tests to be run, in contrast, we determine the order of tests to be investigated after a failed run. Second, to categorize faults as product or environment, we conduct an anomaly analysis. Our list of anomalies is used each morning at the testing team's 10am meeting to determine which faults each developer will investigate.

Is a test failure anomalous? The nature of base station tests makes Ericsson's test environment complex and as a result many tests fail sporadically from environmental problems. Our work has focused on determining the test failure distribution for each test. We label tests that fall outside a 95% confidence interval to be anomalies. Each test has its own failure distribution. For example, tests that fail regularly because of environmental problems will have to fail at a statistically significant higher level to be flagged as a potentially interesting test failure. In contrast, a test that passes every day for two months and fails on a nightly run, will be flagged as an anomaly.

The outcome of this work has been very positively received at Ericsson Ottawa. For example, the testing team, which consists of 8 developers who runthrough test failures every day, use the test anomalies as a technique to prioritize failing tests for investigation. Our interactions with these testers has lead to additional enhancements, such as determining the number of times a test has been an anomaly on the scale of a week, month, and for the current release. Proposed work involves conducting user studies of these testers to determine what and how to show them our results and integrate results into TestGuru.

From a research perspective it is important to understand if this technique can be applied in other settings. We plan to analyze the failure distribution from other Ericsson sites in Sweden. At our suggestion, Herzig, at Microsoft Research is running the anomaly detection technique at Microsoft. His findings indicate that anomalous test failures are almost always environmental faults and that can be safely ignored as they will not usually reappear in a subsequent run.

1.3 Fault Localization through Test Logs

Identifying underlying fault of a test failure is challenging. Unlike conducting code review, when developers manually reading over the code to directly identify issues [], a test failure can be a symptom of different underlying faults and one fault may lead to multiple test failures. Therefore, the diagnose of test failures in order to locate the underlying faults can be time consuming []. This is especially true at Ericsson where a test may fail one day due to an environment problem and another day from a product fault. Although our anomaly analysis helps order the faults and gives a sense of how long a test has been failing, it does not help provide a precise reason and location of the fault, e.g., whether the test is failed due to a newly introduced fault. To assist practitioners of Ericsson in locating test faults, we plan to conduct log analysis on tests. Our co-investigator Dr. Shang is an expert in log analysis. We currently have a masters student, Amar, who is partially funded by a MITACs and has started to collect and clean the logs.

During our ongoing collaboration with Ericsson, we find three major challenges in leveraging logs in Ericsson's testing environments: 1) there is an overwhelming amount of detailed log information, 2) the complex testing environments and context information brings noise into the logs and 3) the complex testing scenarios brings extra varieties into logs. Therefore, we plan our approach by considering address such two challenges.

1. Data collection: We plan to extract all available the test data including a stream of eventsTODO: not sure what is this, counters, and logs. In order to assist in understanding the complex testing environments, we extract the units from the data, *i.e.* which nodes are connected to which mobile devices. TODO: I don't follow this one, are we doing grouping? or just label the testing node and target mobile device?

2. Log abstraction: We plan to abstract logs in order to reduce the context noise. Logs contain static and dynamic information. The static information is specific to each particular event while the dynamic values of the logs describe the event context. We plan to abstract logs into events by vocabulary analysis. The words that exist in the vocabulary of logs but not in the vocabulary of source code are considered dynamic values. By anonymizing such dynamic values, we can abstract logs in to events without context noise.

3. Event sequence generation: Due to the large amounts of logs, understanding system behaviour by reading individual log lines is time consuming. We plan to generate event sequences in order to illus-

trate the execution flow of the tests. Due to the complex testing scenarios, such event sequences are often of great variability even for a similar execution flow. Therefore, we compress event sequences based on testing units to extract generalized event sequences and to remove irrelevant and redundant event sequences.

4. Event sequences comparison: We compare event sequences and we leverage the delta of event sequence to localize test faults. Our comparison consists of two folds: 1) we compare event sequences from runs that are successful with those that are unsuccessful to identify potential problematic events and 2) we compare event sequences from current tests and prior tests to identify whether the tests are failed due to different faults.

Our first novel contribution is to combine both run-time system behaviour during test (i.e., logs) and test results (i.e., pass or failure of the test) to provide actionable suggestions, such as fault localization, to practitioners. Similar approaches have been successful applied by Shang [**?**] to compare the logs from test and production environments. In Ericsson's context, we do not have the production logs, so we must change our comparisons to involve previous test logs from other releases and successful and unsuccessful runs. Secondly, we consider the run-time system behaviour during test as a historical repository. Such system behaviour consists of rich information of the system during execution. We believe the knowledge from learning and analyzing the historical behaviour of the system are of huge benefits to software development and operation.

2 TestGuru: Risk and History

Our work with Ericsson has taken applied problems and developed research-based solutions. To ensure that our results are acted upon, we plan to integrate our results into TestGuru. TestGuru will have three parts. TODO: Emad please elaborate

Localization of risky code: Shihab [] has developed a tool, called CommitGuru, that feeds factors extracted from the software changes and files that are part of the change to determine the likelihood of a change introducing a future defect *i.e.* how risky is the change. CommitGuru uses a logistic regression model, that is trained on historical data, to calculate the probabilities of risk for each change. The factors Features include the number of source code lines changes, the expertise of the author, and the diffusion/dispersion of the change.

A major drawback of the current implementation of CommitGuru is that it only focuses on risk changes, and does not link help with the localization of such changes.

As part of the this proposal, our goal is to add historical test executions and test log information to assess the risk of a change. While some preliminary works have investigated adding these features to statistical fault model[**?**, **?**], our research outcome will be an systematic empirical study of multiple projects to determine how effective tests are at predicting change risk.

Test prioritization: Instead of predicting change risk, we can use the same predictors to determine which are the most effective tests. This will build on our first milestone that is limited to co-failure distribution and add information such as developer expertise. TODO: Could make this more interesting by suggesting the next test to be run instead of prioritizing upfront Test re-ordering

Test creation: Ericsson managers provided examples where there are many tests execution that pass, but customers still find defects. Our models will be helpful in indicating the areas of the system that need critically need new test or need tests to be re-written.

TODO: I have other sections like HQP, but let's just focus on the milestones for now.