Improving regression testing transparency and efficiency with history-based prioritization – an industrial case study

Engström, Emelie; Runeson, Per; Ljung, Andreas

Published in:
[Host publication title missing]

2011

Improving Regression Testing Transparency and Efficiency with History-Based Prioritization – an Industrial Case Study

Emelie Engström*, Per Runeson*† and Andreas Ljung†
*Software Engineering Research Group
Dept. of Computer Science, Lund University, Sweden
(emelie.engstrom, per.runeson)@cs.lth.se
†Sony Ericsson Mobile Communications, Sweden

***Abstract*—*Background*: History based regression testing was proposed as a basis for automating regression test selection, for the purpose of improving transparency and test efficiency, at the function test level in a large scale software development organization. *Aim*: The study aims at investigating the current manual regression testing process as well as adopting, implementing and evaluating the effect of the proposed method. *Method*: A case study was launched including: identification of important factors for prioritization and selection of test cases, implementation of the method, and a quantitative and qualitative evaluation. *Results*: 10 different factors, of which two are history-based, are identified as important for selection. Most of the information needed is available in the test management and error reporting systems while some is embedded in the process. Transparency is increased through a semi-automated method. Our quantitative evaluation indicates a possibility to improve efficiency, while the qualitative evaluation supports the general principles of history-based testing but suggests changes in implementation details.**

***Keywords*-regression testing; history-based prioritization; regression test selection; regression test prioritization; empirical evaluation; industrial case study; function testing;**

I. Introduction

Regression testing (RT) is retesting of previously working software after a change to ensure that unchanged software is still functioning as before the change. The concept of regression testing has changed from being the final gate check before delivery, to being a continuous activity during iterative development. Regression testing is a resource consuming activity in most industrial projects. Studies indicate that 80% of testing cost is regression testing and more than 50% of software maintenance cost is related to testing [1]. The need for effective strategies for regression testing increases with the increasing use of iterative development strategies and systematic reuse across software projects.

A common approach for RT in industry is to reuse a selection of test cases designed by testers [2]. This selection may be static (e.g. determined by the assessed risk and importance of functionality) or dynamic based on impact of changes. In many cases static and dynamic selections are combined and achieved by prioritizing test cases, of which as many are run as admitted by time and resource constraints. As this procedure is based on individual's experience and judgment, it is not transparent enough to enable a consistent assessment of the extent and quality of regression testing. Second, there is no direct evaluation of its efficiency.

There is a gap between research and practice of RT. Even though several systematic approaches for both prioritization and selection of regression test cases are proposed and evaluated in literature [3][4], these are not widely used in industry [2]. This makes the RT highly dependent on people with experience of the product. It is also likely that testers add extra test cases to the scope just to be on the safe side, and thus the testing gets unnecessary costly. There is a need for systematic, transparent strategies for RT selection, which are feasible for an industrial context.

Engström et al. recently reviewed the literature in the field of RT selection techniques [3]. Only few empirical evaluations of regression test selection techniques are carried out in a real industrial context. Yoo and Harman conclude in their recent review of RT minimization, selection and prioritization that empirical studies aggregating the empirical knowledge of RT is "still in the early stages" [4].

We here report on a case study, evaluating history-based regression testing in an industrial setting as a means to improve the transparency of RT selection and prioritization. A prioritization equation proposed by Fazlalizadeh et al. [5] was implemented and used at the function test level in the company. The case study characterizes current practices, proposes and implements an improvement, and evaluates the effects. In this sense, the case study has similarities to action research [6].

The case study is conducted at Sony Ericsson Mobile Communications, a company developing mobile devices with embedded real-time software in a domain which is very competitive both regarding quality and innovation. The development process is highly iterative, and the same software basis is used in several product versions and variants, which all need regression testing. Hence, transparent and efficient RT is important for its contribution to high quality products, efficient use of resources and the lead time for testing.

The paper is structured as follows: Section II overviews related work. The context of our study is described in Section III and the design of the study in Section IV. Results are presented and analyzed in SectionV and a summary of our conclusions is given in Section VI.

II. Related work

Regression testing is a field which is well researched, relatively, within software engineering. Yoo and Harman list 190 papers on RT in their review [4]. Engström et al. identified 36 empirical studies, evaluating 28 techniques for RT selection [3]. Less than a third of the studies comprise industry scale contexts. Since the area is well reviewed recently, we here only focus on the work closely related to the topic under study, namely, empirical evaluations in industry on regression test prioritization and selection.

A. Regression test prioritization and selection

The research on RT distinguishes between three classes of techniques [4]: minimization, selection and prioritization. *Minimization* techniques aim at reducing the test suite by removing redundant and obsolete test cases. *Selection* techniques aim at identifying a subset of a test suite that is sufficient given a certain set of changes. *Prioritization* techniques rank test cases in their predicted order of contribution to e.g. fault detection rate. Test case selection may be applied after test prioritization; a prioritized list of test cases, combined with a cut-off criterion is a selected test suite. Test prioritization can be used in conjunction with test case selection to set the execution order within the selected test suite. This will in turn ensure that if the session is unexpectedly terminated the time spent testing will have been more beneficial than if they were not prioritized [7].

B. Industrial evaluations

The share of industrial evaluations of regression test techniques is low [3]. This is not unique for the RT field, but rather general. Many studies are conducted on the same set of artifacts (the Siemens and the Space programs) which is good from a benchmarking perspective [8], but limits the external validity, since the programs are rather small. Even with industrial artifacts, offline studies tend to reduce the complexity of the real task of test case selection in industry. Published online studies include, for example, White and Robinson [9], Orso et al. [10], Skoglund and Runeson [11], and Engström et al. [12]. See the above mentioned RT reviews for a more comprehensive list [3][4].

C. Factors considered for prioritization and selection

Most selection techniques are change-based [3], i.e. selecting the test suite based on an analysis of the changes from the most recently tested version. Change impact analysis may be conducted at different levels (e.g. statements, modules or files) and based on different artifacts (e.g. code or other system specifications such as UML) or project related information such as error reports. In contrast, most prioritization techniques are not depending on knowledge about modifications [4] but are instead based on supposed equivalents for high fault detection rates (e.g early code coverage, test suite coverage over a number of sessions or historical fault detection effectiveness). Elbaum et al. investigate different coverage criteria that may govern the prioritization [13]. Kim and Porter emphasize the need to view RT as an ordered sequence of test sessions, where the history of the test case executions should be considered [14]. In practice, selection and prioritization are often combined [2], i.e. test cases are prioritized according to some criteria, selected based on some other criteria, and delimited by resource constraints. Engström et al. [12] evaluated a combined technique based on fault proneness of files, originally presented by Kim et al. [15]. Test cases that exercise fault prone files are given higher priority, and selection is based on changed files connected to test cases through fixed error reports. Kim and Porter also conclude that if the test selection technique only focuses on the parts of the program that have been changed since the last testing session, it is a risk that a change, once tested, is never retested, leaving only one chance to find a defect. They proposed a technique that takes this risk into account, and prioritizes test cases based on their history of fault finding rate, its coverage and usage history [14]. Park et al. expanded the model with costs for the execution of each test case [16] and Fazlalizadeh et al. developed this model further by combining the three aspects of Kim and Porter's model into one [5]. Srikanth et al. define a requirements-based prioritization model [17]. It prioritizes test cases based on four factors: customer-assigned priority on requirements, requirement volatility, developer-perceived implementation complexity, and fault proneness of requirements.

D. Black box regression testing

In our context, see Section III, only black box approaches are applicable since the testers do not have full access to the code, and our proposed improvements are based on the ideas from Kim and Porter [14] and are implemented as suggested by Fazlilzadeh et al. [5]. Srikanth et al. claim that most of the test prioritization techniques are code coverage based [17], which is confirmed for selection techniques in the systematic review by Engström et al. [3]. Twenty-six of the 28 identified selection techniques were depending on source code access. The two non-code based techniques are Orso's based on metadata on test case to change information [18], and Sajeev et al.'s UML based approach [19]. In the prioritization area, Qu et al. present a method for prioritizing test cases in a black box testing environment [20] as well as Srikanth et al. [17] as mentioned above. These types of methods require access to various inputs, such as the requirements specifications, customer priority, implementation complexity, fault proneness and other version information metadata. This makes them harder to implement and also research in an offline context.

III. Case Description

At Sony Ericsson the software verification is carried out at different levels by different departments, i.e. unit,

integration, system and acceptance test. The software is divided into different functional areas and for each area there is a group of developers and a group of testers. The integration test is carried out at a test department by a test group for each function. The testing at this level is performed on temporary builds of the software or on the main software branch, usually in a hardware prototype. The system test is carried out by another test department and is performed on the main branch. Finally a release candidate of the software is sent for acceptance test to the carrier service providers.

The software development process is an incremental process, where each component is developed and integrated in small iterations. However, to ensure that the overall quality of the software is maintained, after several iterations, a range of regression test sessions are performed by each function test group on an integrated system. For every new feature, several new test cases are created based on the feature requirements. All the test cases are written for black box testing and are not connected to a specific part of the code. The test cases are added to the test database which contains all the test cases relevant to a specific product. The amount of features increases in each project and therefore the total amount of test cases available is too large to re-test all during RT on a regular basis.

All test case descriptions and the execution data are stored in a commercial tool, HP's Quality Center (QC). QC is a web based test database that supports essential aspects of test management. The defect reports are stored in a defect management system (DMS) and linked via an id number, to the revealing test case in QC and also to other test cases that are affected by the defect. Sony Ericsson uses QC as its main test management tool. Test planning, test design and test execution is performed in the QC environment. Each executed test case should contain the following information: Software version, hardware version, test status (Blocked, Failed, N/A, No Run, Not Completed, Passed), execution date and time, tester id and if the status is Failed a DMS number from the defect report in the DMS.

IV. Case study design

The design of this case study is outlined below in line with the guidelines by Runeson and Höst [6].

A. Objective

The objective of the study is to improve regression testing at function test level by adapting and implementing history-based regression testing into the current context. In this case improvements refer to increased transparency of the test scope selection procedure as well as increased, or at least maintained, test effectiveness. With an automated selection procedure (history-based selection) both goals are expected to be achieved. Thus we wanted to investigate current practices in order to identify a proper level and type of automation and evaluate the effects of implementing it.

B. Research questions

The research questions for the study are the following:

1) Which factors are considered when prioritizing and selecting test cases for regression test?
2) Do history-based prioritization and selection of test cases improve regression testing?

C. Case and unit of analysis

The case under study is the regression testing activities in an iterative, incremental development process for complex, large-scale software development. The unit of analysis in the study is specifically regression testing carried out by one function group at Sony Ericsson.

D. Procedure

The case study is carried out in several steps, starting with A) exploratory semi-structured interviews with the purpose of identifying important factors for test case prioritization and selection, and comparing current practices and expert opinions with literature on history-based testing. The next step was to B) select and implement a suitable method. The prioritization technique proposed by Fazlalizadeh et al. [5] was implemented in two versions: one as close to the original as admitted by the context and one including extensions suggested in the exploratory part. The methods were C) quantitatively evaluated with respect to their fault detection efficiency and finally D) the testers' opinions about the implemented methods was collected and analyzed.

V. Case study report

A. Exploring current practices

Semi-structured interviews were held with one test engineer and one technical project leader in the function test group. Areas discussed were the current process and test selection strategy, problems and strengths with current practices, their confidence in the selected suites, factors considered important for prioritization and their opinions about history-based regression testing as proposed by Fazlalizadeh et al. [5]. The main topics of the interviews are listed in Table I.

In addition to the interviews, observations made by the third author of this paper (after several months of active participation in the work of the group) were taken into account. The general description of the work practices is reported as a case description in Section III. Experienced problems with the current regression testing method, and thus the expected benefits of automating selection and prioritization of test cases, are discussed below. Factors identified in the interviews which were considered important for regression testing are presented in Tables II and III, and issues related to the practical implementation of these factors are discussed in Section V-B. Table II presents a list of factors for selection of test cases. Other factors and properties were identified

Table II
FACTORS IDENTIFIED FROM THE INTERVIEW THAT SHOULD AFFECT THE TEST CASE PRIORITIZATION EQUATION (IN NO SPECIFIC ORDER). FACTORS ADDED IN THE EXTENDED APPROACH ARE MARKED WITH AN ASTERISK (*)

Factor	Rationale
Historical effectiveness	*The defect detection frequency of the test case during a period of time.* This is a measure of the test case's effectiveness. If a test case often reveal defects it may indicate that it exercises parts of the software where new defects often appear. A test case that detects a defect is linked to the defect report in the defect management database.
Execution history	*The number of executed regression test sessions since the latest execution of the test case.*It is important to ensure that all test cases eventually are executed over a period of time in round Robin fashion. So the number of sessions that has been executed without the test case should increases the test case's priority until it is executed in a regression test session.
Static priority*	*The importance of the test case, for business priorities and for the overall functionality.* This aspect should be incorporated in order to ensure that some important basic test cases are given higher priority. This property is set manually when the test case is created and should affect the priority in every selection.
Age*	*The creation date of the test case.* This aspect should be incorporated in order to ensure that new functions are more thoroughly tested. The creation date may be used to determine which test cases are new.

Table III
FACTORS IDENTIFIED FROM THE INTERVIEW THAT WOULD MOTIVATE EXCLUSION OF TEST CASES. IMPLEMENTED FACTORS ARE MARKED WITH AN ASTERISK (*)

Factor	Rationale
Cost	*The cost of a test case.* This property should be used to estimate the execution time for each test case. An automated test case is assumed to consume less time and resources.
Focus of session*	*The test case type (for example Duration test, Performance test, Certification).*This property should be used to exclude test cases that are marked for duration, performance and certification that should not be executed in a regression test session with focus on functional tests.
Scope of session *	*If the test case is written for a specific hardware that is not available in all products.* This property should be used to exclude test cases that are not applicable for the product about to be tested.
Redundant test cases	*Current Status of the test case.* This property should be used to exclude test cases revealing already revealed defects.

Table I
QUESTIONS FOR THE INTERVIEWS OF THE TEST ENGINEER (TE) AND TECHNICAL PROJECT LEADER (TPL).

Question	TE	TPL
Define the development and test process. When is regression testing performed?		X
How do you select the test cases to be run during a regression test?	X	
How do you evaluate how much time each test case will add to the whole test suite?	X	
What differentiates the test cases from your point of view?	X	
Do you feel confident that the correct suite has been selected every time?	X	
What is the most important factor when running the selected test suite: time/resources or quality?	X	
Which properties of the executed test cases should affect the prioritization equation in the test case selection technique?	X	
What do you think of the factors in the history-based method?	X	X

that would motivate de-selection of certain test cases; these are listed in Table III.

One of the problems with the current method is that it depends on experienced testers with knowledge about the system and the test cases. There is a risk that the selected test suite is too extensive or too narrow; a tester with lack of experience in the area could have trouble estimating the required time and resources. Moreover, the selected test suite may be inefficient and misleading, since it is just based on judgment. The tester has to know which test cases, recently, have been more prone to detect faults. Another problem is that even an experienced tester could select an inefficient test suite. The test cases are selected in a routinely manner by just selecting the same test cases for every regression test session, and since the selection is based on judgment, there is no evidence that it is the most efficient test suite. Hence, the following are the expected benefits of a tool supported selection method:

- increased transparency
- improved cost estimations
- increased test efficiency
- increased confidence

These findings motivates tool support and are a basis for decisions on how to implement and evaluate it. *Increased transparency* is achieved with any kind of automation, since no systematic method for regression testing is currently used. If the use of a systematic method or implemented tool does not decrease test efficiency or confidence we consider it an improvement of the current situation. No data regarding the *execution cost* for a test case or group of test cases was available and thus this aspect could not be evaluated

within the scope of this case study. *Test efficiency* regards the number of faults revealed per executed test case and is evaluated in two ways in this case study: 1) by comparing the efficiency of the execution order (prioritization) of test cases in the suites and 2) by analyzing test suite selections of the same magnitude as corresponding manually selected suites. To reach and measure *confidence* in a method is in itself non-transparent, since it deals with the gut feelings of the testers. To some extent it relates to coverage. If a method can be shown to include all important test cases, the confidence in it is high. However, optimizing a method to include important coverage aspects affect test efficiency negatively, since it adds test cases to the selection without respect to their probability of detecting faults.

B. Implementation of history-based testing

An analysis of the identified problems together with an overview of the research on regression test case selection and prioritization, led to the hypotheses that a semi-automatic test case selection tool, based on a history-based test case prioritization technique offers a solution for the problems in this verification process. The hypothesis that history-based regression test selection could improve the current situation was a starting point for this case study and it was further supported in the exploratory step. The technique proposed by Fazlalizadeh et al. [5] was selected for implementation because it covers some of the desired aspects identified in the exploratory step. For information about other history-based techniques see Section II.

However, the proposed equation could not be implemented without adaptation since it requires information not available within this environment. This implementation is referred to as the *Faz* approach. An extended version was implemented for evaluation as well, referred to as the *ExtFaz* approach. The technique was extended with both selection criteria and prioritization criteria.

1) The Faz approach: The strategy proposed by Fazlalizadeh et al. [5] is based on historical performance data and incorporates three factors: historical effectiveness in fault detection, each test case's execution history in regression test and the last priority assigned to the test case. Priorities are calculated according to the following formula[1]:

$$PR_k = \alpha * f_{ck}/e_{ck} + \beta * PR_{k-1} + \gamma * h_k$$
$$0 <= \alpha, \beta, \gamma <= 1, k >= 1$$

The equation consists of the following parts:

- *Historical effectiveness* (f_{ck}/e_{ck}): is the number of times the test case has failed and e_{ck} is the number of test case executions during k number of test sessions.
- *Execution history* (h_k): Each time a test case is not executed, its execution history will be increased by one. Once the test case is executed, execution history becomes 0 and the operation is repeated.
- *Previous priority* (PR_{k-1}): is the latest priority of the test case. The initial priority, PR_0, is defined as the percentage of code coverage of the test case.
- *Three weighting parameters* (α, β and γ): To balance the effects of the factors, the parameters α, β and γ can be changed. The values are always between 0 and 1.

The prioritization equation described above includes two of the desired factors listed in Table II: 1) the defect detection frequency and 2) the number of regression test sessions that has been executed since a test case was last executed.

Most of the information needed to apply this equation is available in the test management database. However, information about code coverage is not available so the initial prioritization had to be based on something else than code coverage. We chose to use the static prioritization assigned to each test case at creation. The PR_0 value is selected so that it will add a proportional amount to the total priority sum and the term ($\beta * PR_0$) will be between 0 and 1. The test cases are given the following PR_0 value:

$$PR_0 = \begin{cases} 0.4 & \text{if priority is 1} \\ 0.2 & \text{if priority is 2} \\ 0.1 & \text{if priority is 3} \end{cases}$$

2) The ExtFaz approach: In an attempt to incorporate the other desired factors, an extended version of the approach was developed. Two more factors from Table II were added as a basis for prioritization and selection of test cases: 1) the static priority and 2) the age of a test case. PR_0 is here used both as a value for the initial priority and for the continuous calculated priority, giving more weight to static priority when the test case is new. These two properties are added separately to the priority value for each calculation. The extended equation is defined as follows:

$$CalculatedPriority_k = PR_k + N_k + I_k$$

where I_k is defined as PR_0 above, and

$$N_k = \begin{cases} 0.4 & \text{if current date – creation date} < 3 \text{ months} \\ 0 & \text{if current date – creation date} > 3 \text{ months} \end{cases}$$

- *Static priority:* The test case's original priority should affect the calculated priority since the most important test cases always should be given higher priority. The value is chosen with the same relation between the priorities as in PR_0 but with the same magnitude as PR_k which is between 0 and 1
- *Age:*The creation date of the test case reveals new test cases and ensure that they are given higher priority. The limit for how long a test case is defined as new is set to 3 months. The value is chosen so that it is of the same magnitude as PR_k.

[1]In the original paper the PR_{k-1} on the right hand side of the equation is actually presented as PR_k but this is assumed to be a typing error.

Table IV
NUMBER OF EXECUTED TEST CASES AND FAULTS FOUND FOR EACH VERSION, USING THE ORIGINAL EXPERIENCE-BASED METHOD.

Version	R1.0	R1.1	R1.2	R1.3	R1.4	R1.5
Executed	350	351	450	436	436	435
Faults	4	7	5	8	13	8

The following factors were considered important but were due to different reasons not possible to include:

- *Redundant test cases:* To avoid selecting a test case that reveal a defect that has not been fixed yet the link to the defect report should be checked. If the defect report linked to the test case still is unresolved the test case should have low priority. This link however could not be set up during the development of the prototype tool.
- *Cost:* There is no data about how much time and resources either of the types manual or automatic test cases will consume, therefore this property is not included into the equation.

A prototype tool was developed for the evaluation. All test cases are stored in an SQL database on the Quality Center server and the application collects data from the data base with SQL queries. Both equations were implemented in one tool with a user interface allowing for some choices: which equation to use, the constraints on the session (i.e. number of test cases), exclusion of certain types of test cases (e.g. performance, duration or certification tests). The variant specific test cases could also be excluded but this information is often specified only in the test case's description text and could not be retrieved. However, in some cases this information could be found in the name of the test cases and these test cases are possible to exclude from the prioritization equation.

C. Comparative evaluation of strategies (Exp, Faz, ExtFaz)

The three strategies, 1) current experience based (Exp), 2) the proposed strategy [5] (Faz) and 3) the extended version of the proposed strategy (ExtFaz) were compared through a quasi-experiment [21]. Data were collected from six recent consecutive regression test sessions of six consecutive software versions, and the effect of the prioritization of execution order and selection with the tool were analyzed and compared with the actual outcome of respective session. Execution status (pass/fail) were known only for test cases actually executed in the original sessions.

The total number of test cases in the pool is 2 114. The numbers of executed test cases and the number of faults found in each session are presented in Table IV and provides raw data for the evaluations described in this section.

1) Evaluation of execution order: The manually selected experience-based set of test cases (which is the *total number of executed test cases* in Figure 3) was prioritized according to the two proposed prioritization equations. The prioritized suites were then compared with respect to their ability to detect faults early. A metric introduced by Rothermel et al. [7] was used for this purpose: APFD, measuring the weighted *average of the percentage of faults detected* over the test session. A high value means that the total number of detected faults becomes high early in the session. The APFD for a test suite is given by:

$$APFD = 1 - \frac{TF_1 + TF_2 + ... + TF_m}{nm} + \frac{1}{2n}$$

where TF_k is the rank of the test case, detecting the k:th fault, n is the number of executed test cases and m is the number of detected faults. See Rothermel et al. [7] for more details.

The APFD values for each of the six sessions is reported in Figure 1. Here the history-based techniques perform better in all six sessions. This means that if the selected test cases are executed in a prioritized order as recommended by the tool, faults would be revealed earlier. However, since test suites are executed in relatively short sessions, this is not a very important improvement in itself. A more important consequence is that if testing is interrupted and not all test cases are executed, the prioritized order guarantees the highest achieved fault detection efficiency at any time. Note that this measure does not say anything about the effect of the manual selections since the execution order is not a basis for the manual selection.

The worst case scenario, with respect to our assumptions, in comparing APFD values is reported in Figure 2. If the

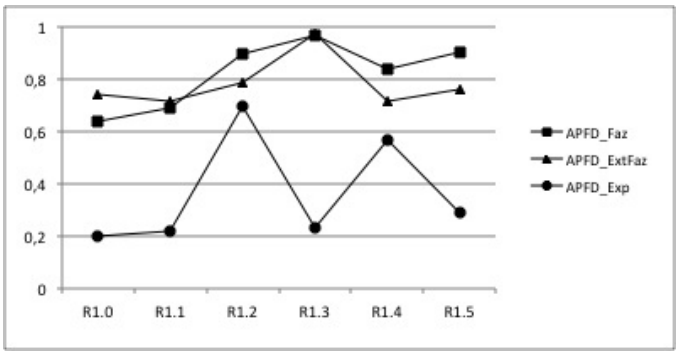

Figure 1. APFD-values for each version.

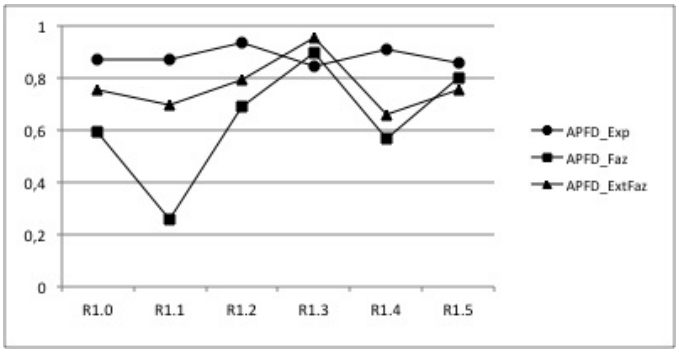

Figure 2. Worst case scenario on comparing APFD values.

Table V
SIZE CONSTRAINTS ON SELECTION (# TEST CASES) IN EACH VERSION AND THE SHARE OF KNOWN VERDICTS (# SELECTED AND EXECUTED TEST CASES) FOR EACH METHOD

Version	R1.0	R1.1	R1.2	R1.3	R1.4	R1.5
Selected	350	351	450	436	436	435
execFaz	279	95	173	64	86	75
execExtFaz	80	131	207	66	136	138

actual selection of test cases were safe (i.e. including all fault revealing test cases) the pure history-based technique (Faz) performs significantly worse than the other two approaches in the first three sessions, and as good as the extended version (ExtFaz) in the last three sessions, while the manual approach would have very high APFD values in all sessions. This is not very surprising since the assumption is that the percentage rate reaches 100% within the selected scope. The testers' uncertainty regarding their selections indicates however that this is very unlikely.

2) Evaluation of selection: For each session we also used the tool for selecting a set of test cases recommended for execution. The constraints for the selection was set to match the size of the original selection. The selected suites were then compared with respect to fault detection efficiency defined as:

$$Eff_{det} = \frac{\#\ faults\ found}{\#\ test\ cases\ executed}$$

However, we only have access to execution data from the originally executed test suite. Table V shows the size of the selection as well as the number of known verdicts for the two tool based selections (i.e. the number of *executed selected test cases*, see Figure 3). Only the executed share of the selected test cases are available for evaluation and some assumptions about the *non-executed selected test cases* are necessary. The extremes are of course that they are all false positives (pass), or all true positives (fails). In Table VI the number of faults detected are reported for the two methods.

In Figure 4 we report on the efficiency of the evaluated methods as an effect of different assumptions and compare with the actual efficiency for each version. In any case the efficiency is rather low, between 1 and 10%, and there is room

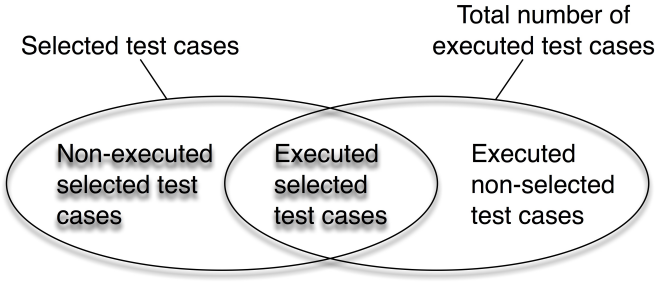

Figure 3. Diagram relating the sets of test cases to each other.

Table VI
NUMBER OF FAULTS FOUND FOR EACH VERSION BY THE TEST CASES SELECTED BY EACH METHOD

Version	R1.0	R1.1	R1.2	R1.3	R1.4	R1.5
Known faults	4	7	5	8	13	8
Faz	2	2	3	6	6	6
ExtFaz	2	2	2	7	7	6

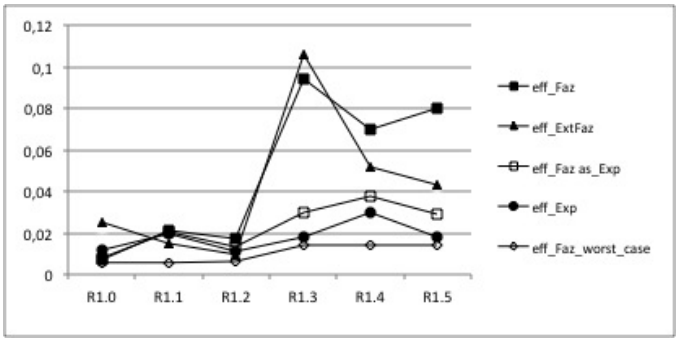

Figure 4. Sensitivity analysis for each version, showing the efficiency for the three approaches and two calculations based on different assumptions.

for improvements. The efficiency analysis of *eff_Faz* and *eff_ExtFaz* is based on executed and selected test cases only, see Figure 3, and thus the assumption is that the efficiency measured is representative for the non-executed test cases. This analysis compares the two automated methods. The two techniques only differ in efficiency for the last version. For further comparisons the Faz technique represents the tool, *eff_Exp* is based on the total number of executed test cases. The *eff_Faz_as_Exp* assumes that the non-executed selected test cases is as efficient as in the manual case, while *eff_Faz_worst_case* assumes that no defect is found by the non-executed selected test cases.

We interpret this as the Faz method has a potential to maintain or even improve the efficiency. The worst case scenario may be worse than he manual method, but this scenario is very unlikely.

The first three versions show no big differences independent of the assumption while the last three versions widen the range of possible efficiencies. This is natural, since there is an embedded learning property of the the history-based techniques.

D. Expert opinions about the resulting prioritization

A test suite, automatically selected with the ExtFaz prioritization technique, was handed out to five testers in the team. The test suite was presented in two different orders in Excel sheets, one where the test cases were grouped by function to allow the tester to easily review his/her part of the test suite and one where the test cases were presented in priority order to allow the testers to review the proposed priority order. They were then given a form to fill out and give their opinion about the test suite and the priority order

Table VII
EVALUATION FORM HANDED OUT TO THE TESTERS. THE STATEMENTS WERE GRADED ON A FIVE STEP LIKERT SCALE : STRONGLY DISAGREE, SLIGHTLY DISAGREE, DON'T KNOW, SLIGHTLY AGREE, STRONGLY AGREE.

1. The automatically selected test case selection is equal to what I would have selected for this session.
2. The automatically selected test case selection contains test cases that are inappropriate (unnecessary).
3. The automatically selected test case selection is missing important test cases.
4. The automatically selected test case selection would ensure that the quality demands are met.
5. The automatically selected test case selection could be executed within the time and resource limit of an ordinary regression test session.
6. The priority list matches my expectations, this is the same way I would have prioritized the test cases (generally).

of the test cases. The form contained six statements see Table VII and also had space for two open questions.

The testers were positive to automating the selection of test cases and believed a history-based approach could increase test efficiency. However, several problems with this specific implementation were identified. Most of the testers stated that the automatically selected test suite included unnecessary test cases. A reason for this is lacking or outdated information in the test management database. Lack of configuration information led to many non-relevant variant specific test cases. The static priority is not updated as test cases' importance decrease. Another reason for including unimportant test cases was incorrect assumptions in the priority equation. Execution history was not a good surrogate for fault detection capabilities nor was the age of a test case.

The link to the defect reports was requested by most of the testers. It would solve some problems. For example, blocked test cases (with open issues) could be excluded from the test suite, and test cases which had been blocked in a previous session should be considered as fault detecting in that session. It is also relevant to consider the severity of the detected faults. One tester noticed that test cases detecting faults of low severity were given high priorities, since these faults were not fixed and the test cases had not been selected for execution since.

One tester asked for a possibility to prioritize between different groups of functions, rather than individual functions in order to reach a better coverage of function areas.

E. Threats to validity

This study comprises several steps, combining two research methodologies: the exploratory case study and the evaluative quasi-experiment. Threats to validity depend on type of study and goals and are here analyzed from both perspectives according to the following taxonomy [6], [21]: construct validity, internal validity, external validity and reliability.

1) construct validity: Construct validity refers to whether the design of the study represents a fair investigation of the research questions. There are several threats to validity here: the selection of the case, the interpretation of questions and answers in interviews, the use of proper metrics and experimental setups for evaluation. Since the primary goal of the study was to improve the current situation in this context the selection of case is trivial. Our case represents a non-trivial real life situation and could as such be regarded as a typical case [6]. However, there are many variation factors in a real life regression test situation [2] and it is not possible to find a commonly accepted typical case. Thus we do not claim to have a general solution.

The metrics used for comparing the techniques are accepted by the research community and have been used in several previous studies for the same purposes [22], [7], [13]. There is however other views on what is a good selection (e.g. high inclusiveness or precision [22]) or prioritization (e.g. early code coverage) of test cases which we could not analyze in this study due to limitations of available data. Concerns about the experimental setup regard the fact that only a subset of the test cases were executed and assigned an execution status. Thus assumptions about the non-executed test cases have to be made. Since the set of executed test cases does not represent a randomized sample any such assumption is biased. As a countermeasure to this threat, a sensitivity analysis is conducted, where the effect of different assumptions are reported and boundaries for worst cases are identified.

2) internal validity: Internal validity refers to whether the interpretation of the results is correct. This threat does not apply to the exploratory part of the study since no casual relationships are studied. Instead threats to internal validity concerns the analysis of data in the evaluative part. The implementation of the history-based testing is tailored for this specific context. Thus we do not draw conclusions about the specific technique selected for implementation but rather about the general concept of history-based testing and about the possibility of improving regression testing with systematic automatable strategies. There may by unknown factors behind the selections in the benchmarking regression test sessions making the sets of actually executed test cases bad representatives. Interviews before implementation and experts' opinions about the recommendations are countermeasures to this threat.

3) external validity: External validity refers to whether the findings are possible to generalize. Analytical generalization [6] is supported through a thorough description of the case, see Section III. Statistical generalization is not possible from a single case study but the concepts need to be evaluated further in different contexts.

4) reliability: Reliability refers to whether the study is conducted in a robust manner and can be repeated by other researchers with the same results. There is a major threat in

the exploratory part of this study. Interviews are only semi-structured and not recorded so it is possible that another researcher would have identified a different list of important prioritization factors. However the list proposed in this study is further evaluated and thus validated within the scope of this study. Another threat is the implementation of the prototype tool. Validation of the implementation was made by another researcher reviewing the code.

VI. Conclusion

In this paper we report on a case study of the implementation of history-based regression testing for the purpose of improving transparency and test efficiency at function test level in a large software development organization. Current practices are investigated and a semi-automated tool, combining the concepts of history-based regression testing in literature with good practices in the current process as well as practitioners' opinions, are implemented. Three different strategies, the current experienced based approach and two systematic approaches are empirically compared through a post hoc quasi experiment. The outcome of the tool is further assessed through manual reviews made by the practitioners.

History-based prioritization do account for properties of previous executions of test cases in order to increase test efficiency of a test suite. Two such properties were initially identified as important to incorporate into a tool by the practitioners: historical effectiveness and execution history. However, after implementation and evaluation of our tool, execution history was discarded as a basis for prioritization by the practitioners. The non-historical factors identified aim at increasing confidence rather than efficiency of testing. A history-based method proposed by Fazlalizadeh et al. [5] was considered a good basis for prioritizing test cases, while not covering all of the important factors. It is intuitive and the historical data needed for analysis is available in the test management system.

We conclude regarding the two research questions defined in Section IV:

RQ1. Which factors are considered when prioritizing and selecting test cases for regression test? Following factors are considered important and thus affect the manual prioritization and selection of regression test cases: *Historical effectiveness, Execution history, Age* and *Static priority* of a test case affect which priority a test case is given, while *Scope* and *Focus* of the session as well as the session's *Time and resource constraints* is a basis for selection of test cases. Test cases are prioritized not only individually but also with respect to which function areas they belong to in order to achieve a reasonable *Coverage of function areas*. *Current status* (e.g. blocked or postponed) of a test case is considered for filtering out invalid test cases while its *Status* in previous sessions affects the current priority.

RQ2. Do history-based prioritization and selection of test cases improve regression testing? Transparency is improved through automating as much as possible of the selection procedure. Such automation should incorporate context specific factors as well as good principles accepted in research literature. However details in proposed techniques are less important since adaptation to the current process, available tools and practices are inevitable. It is not possible to cover every aspect of the selection procedure in a tool and thus guidelines for how to use the tool are needed as well. The tool may be more useful in some regression test sessions than others (e.g. if for example the focus is on problem areas). Even with a tool, the selected test suites should be manually reviewed both for the purpose of identifying gaps and gaining knowledge of important test cases.

Our quantitative evaluation shows that history-based prioritization of an already selected test suite improves the ability to detect faults early, which is good if a test session is prematurely interrupted. More important is the efficiency of a selection which is based on the total prioritization. Our data does not admit an exact evaluation of this but indicates an increase in fault detection efficiency, while a worst case interpretation of available data shows a small decrease in efficiency.

The history-based method was extended with components accounting for two additional factors for which data was available in the test management system, the age and static prioritization of test cases. These extensions did increase the testers' confidence in the test suites but they did not affect the efficiency of neither the prioritization nor the selection.

In summary, testers were positive to the general ideas of history-based prioritization and of incorporating good practices into a tool, but they were not satisfied with the details of this specific implementation. Many of the problems encountered can be resolved by changes in the implementation. Some of the not implemented important factors could be incorporated with a link to the error reporting system. However, some shortcomings of the tool relate to outdated or missing information in the test management database and cannot be resolved as easily.

Acknowledgements

The work was partly funded by Vinnova under grant 2007-03005 for the SWELL research shool, and partly by the Swedish Research Council under grant 622-2007-8028 for a senior researcher position in software engineering. The authors are thankful to Bruno Einarsson, Roger Hassel and Hares Mawlayi at Sony Ericsson for their support during the project.

REFERENCES

[1] P. K. Chittimalli and M. J. Harrold, "Recomputing coverage information to assist regression testing," *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 452–469, 2009.

[2] E. Engström and P. Runeson, "A qualitative survey of regression testing practices," in *The 11th International Conference on Product Focused Software Development and Process Improvement*, 2010, pp. 3–16.

[3] E. Engström, P. Runeson, and M. Skoglund, "A systematic review on regression test selection techniques," *Information and Software Technology*, vol. 52, no. 1, pp. 14–30, 2010.

[4] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, pp. n/a–n/a, 2010. [Online]. Available: http://dx.doi.org/10.1002/stvr.430

[5] Y. Fazlalizadeh, A. Khalilian, M. Azgomi, and S. Parsa, "Prioritizing test cases for resource constraint environments using historical test case performance data," *2009 2nd IEEE International Conference on Computer Science and Information Technology*, pp. 190–195, 2009.

[6] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[7] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, pp. 929–948, 2001.

[8] P. Runeson, M. Skoglund, and E. Engström, "Test benchmarks – what is the question?" in *First Software Testing Benchmark Workshop TESTBENCH'08 Co-located with ICST 2008 First International Conference on Software Testing, Verification and Validation*, M. Roper, Ed., 2008.

[9] L. White and B. Robinson, "Industrial real-time regression testing and analysis using firewalls," in *Proceedings 20th IEEE International Conference on Software Maintenance*, 2004, pp. 18–27.

[10] A. Orso, N. Shi, and M. J. Harrold, "Scaling regression testing to large software systems," in *SIGSOFT '04/FSE-12: Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering*. New York, NY, USA: ACM, 2004, pp. 241–251.

[11] M. Skoglund and P. Runeson, "A case study of the class firewall regression test selection technique on a large scale distributed software system," in *International Symposium on Empirical Software Engineering*, 2005, pp. 72–81.

[12] E. Engström, P. Runeson, and G. Wikstrand, "An empirical evaluation of regression testing based on fix-cache recommendations," in *Proceedings of the 3rd International Conference on Software Testing Verification and Validation*, 2010, pp. 75–78.

[13] S. Elbaum, G. Rothermel, S. Kanduri, and A. G. Malishevsky, "Selecting a cost-effective test case prioritization technique," *Software Quality Journal*, vol. 12, no. 3, pp. 185–210, 2004.

[14] J.-M. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," in *Proceedings of the 24rd International Conference on Software Engineering*, 2002, pp. 119–129.

[15] S. Kim, T. Zimmermann, E. J. Whitehead Jr., and A. Zeller, "Predicting faults from cached history," in *ICSE '07: Proceedings of the 29th international conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 489–498.

[16] H. Park, H. Ryu, and J. Baik, "Historical value-based approach for cost-cognizant test case prioritization to improve the effectiveness of regression testing," in *Second International Conference on Secure System Integration and Reliability Improvement*, 2008, pp. 39–46.

[17] H. Srikanth, L. Williams, and J. Osborne, "System test case prioritization of new and regression test cases," in *International Symposium on Empirical Software Engineering*. Los Alamitos, CA, USA: IEEE Computer Society, 2005, pp. 62–71.

[18] A. Orso, H. Do, G. Rothermel, M. J. Harrold, and D. S. Rosenblum, "Using component metadata to regression test component-based software," *Software Testing, Verification and Reliability*, vol. 17, no. 2, pp. 61–94, 2007.

[19] A. Sajeev and B. Wibowo, "Regression test selection based on version changes of components," in *Tenth Asia-Pacific Software Engineering Conference*, 2003, pp. 78–85.

[20] B. Qu, C. Nie, B. Xu, and X. Zhang, "Test case prioritization for black box testing," in *Annual International Computer Software and Applications Conference*, vol. 1. Los Alamitos, CA, USA: IEEE Computer Society, 2007, pp. 465–474.

[21] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, 2000.

[22] G. Rothermel and M. J. Harrold, "Analyzing regression test selection techniques," *IEEE Transactions on Software Engineering*, vol. 22, no. 8, pp. 529–552, 1996.