Test Prioritization based on Change Sensitivity: an Industrial Case Study

Cu D. Nguyen

Paolo Tonella

Bilha Mendelson

Daniel Citron

Onn Shehory

Tanja E. J. Vos

Nelly Condori-Fernandez

Technical Report UU-CS-2014-012
May 2014

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Test Prioritization based on Change Sensitivity: an Industrial Case Study

Cu D. Nguyen[1], Paolo Tonella[1], Bilha Mendelson[2], Daniel Citron[2], Onn Shehory[2], Tanja Vos[3], and Nelly Condori[3]

[1] Fondazione Bruno Kessler, Trento, Italy
`{cunduy,tonella}@fbk.eu`
[2] IBM Research Lab, Haifa, Israel
`{bilha,citron,onn}@il.ibm.com`
[3] Universidad Politécnica de Valencia, Valencia, Spain
`{tvos,nelly}@pros.upv.es`

Abstract. In the context of service-based systems, applications access software services, either home-built or third-party, to orchestrate their functionality. Since such services evolve independently from the applications, the latter need to be tested to make sure that they work properly with the updated or new services. In a previous work we have proposed a test prioritization approach that ranks test cases based on their sensitivity to external service changes. The idea is to give priority to the tests that detect the highest number of artificial changes (mutations), because they have a higher chance of detecting real changes in external services. In this paper, we apply change-sensitivity based test prioritization to an industrial system from IBM within the FITTEST European project. Results indicate that the ranked test cases achieve automatically comparable performance as manual prioritization made by an experienced team.

Keywords: Test Prioritization, Change Sensitivity

1 Introduction

Web services are widely adopted in many businesses, from social media, banking and e-commerce, to end-user mobile applications. There are several service providers who offer Web services facilities, e.g. Web API, REST[9] or SOAP[1] Web services, allowing service consumers to easily access, integrate, and share their business data. At the time of writing this paper, according to the site Programmable Web[4], there are more than 10 thousands services in the format of Web API available. These numbers are increasing constantly. As a result, service-based systems and architectures are being developed at a similar pace. Many companies, such as IBM, design their software systems such that they orchestrate internal and external services, on top of a service-oriented architecture [6].

However, these services have to evolve quickly to meet technology and business changes. Service providers may optimize the performance (e.g., changing data format from XML to JSON), or they may develop new features to meet new requirements. For instance, Amazon released 15 versions[5] of their *Elastic Compute Cloud* service in 2012. As a result, service integrators face frequently a critical decision either to update their service compositions to the new versions of the services they are exploiting, or to stay with the old ones, knowing that the old services might have issues, risks, and limited support. More often than not, the latter is the preferred choice. Sometimes, the old services are dismissed; hence migration to the new services is mandatory.

Audit testing aims at checking the compliance of a new service, including a new version of an existing service or a newly discovered service from a new provider, with a system under test (SUT) that integrates the service and currently works properly. Audit testing is a form of regression

[4] See `http://www.programmableweb.com`
[5] `http://aws.amazon.com/releasenotes/Amazon-EC2/0470228374296018`

testing [16, 17, 14], since the goal is ensuring that the pre-existing functionalities are preserved when the new versions of the service are used. In audit testing, the role of test prioritization is crucial because the time budget for testing is often limited so that the SUT can be back in operation quickly.

The European project FITTEST (Future Internet Testing) [7] has developed a regression technique that is aimed at automatically prioritizing a set of test cases based on their sensitivity to external changes [14]. The key idea is to give priority to the tests that can detect a high number of artificial changes. Mutation techniques tailored to the inter-service communication are used to generate changes in the messages exchanged between the SUT and the services. These changes are similar to unexpected contents in the service messages that could happen due to service evolution. The test cases that detect such changes have a higher chance of detecting real changes that matter in external services. Therefore, they are ranked higher.

The preliminary results of evaluating these regression testing technqies [14] obtained in an experimental setting show the effectiveness of the approach. In contrast, the study presented in this paper aims to obtain empirical evidence of the applicability of these techniques within the context of an industrial team, testing an industrial system. To this end we present a "which is better" type of case study [11]. These case studies are powerful since, although they cannot achieve the scientific rigour of formal experiments, the results of a case study can provide useful insights to help others judge whether the specific technology being evaluated could benefit their own organization [11, **?**,**?**,**?**,**?**]. In order to assess tools, evaluative case study research must involve realistic systems and realistic subjects, as explicitly done in this study.

The study has been executed at the IBM Research lab in Haifa[6] within a simulated test environment for an industrial system, called IT Management Product, or IMP for short. It is a large-scaled networked system that controls and optimizes resource management, such as creating and reconfiguring virtual machines on demand.

The contribution of this paper is two-fold. On the one hand, it describes promising results and a successful adoption of a research result for regression testing in a real industrial setting. On the other hand, the study in this paper can serve as an example that others can follow when encountering the need to evaluate a (regression) testing tool in an industrial setting following the FITTEST methodology for evaluating testing tools that can be found here [19].

The remainder of the paper is organized as follows: Section 2 provides background on sensitivity based test prioritization for audit testing of services; Section 3 describes the design of our empirical study; Section 4 presents and comments the experimental results; Section 5 concludes the paper.

2 Background

Web service clients communicate with services via message passing: clients send requests to services and receive responses. These messages can use two main different standard formats: XML (for SOAP [1] and RESTFul [15]) and JSON[7]. Both are structured, tree-like formats.

During an audit testing session at the client side (see Figure 1) test inputs are provided to the client system, which then sends requests to services, and receives and processes the responses. Finally, the outputs from the system are evaluated (the test oracle takes usually the form of assertions, the default assertions being that the application should not crash or raise exceptions). Since the client lacks controllability and observability over the service, and SLA (Service Level Agreement) [12] concerns only high-level quality contracts (e.g. performance, response time), the developer of the client (service integrator) has to make assumptions about technical details regarding the format of the responses. We denote these assumptions as *service integrator's assumptions*. For example, the integrator might expect a list of phone numbers, separated by commas, from the

[6] This is a case study conducted by the Research team at IBM and does not necessarily reflect the development and the testing process of IBM products.
[7] http://www.json.org

service, when searching for a phone number. Changes in the data format of the response (e.g., using colon instead of comma as phone number separator) may break the assumptions of the service integrator, which may lead the client to misbehave, thus failing the test cases.

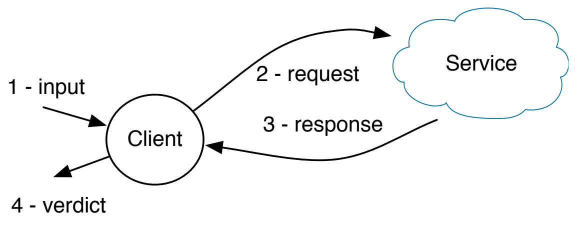

Fig. 1. A test session at a service client

```
<onElement xpath="//title" name="title">
  <restriction base="string">
     <minLength value="5"/>
     <maxLength value="256"/>
   </restriction>
</onElement>
<onElement xpath="//item/conditionId" name="conditionId">
   <restriction base="string">
     <enumeration value="Used"/>
    <enumeration value="New"/>
     <enumeration value="Unspecified"/>
   </restriction>
</onElement>
<onElement xpath="//timeLeft" name="timeLeft">
   <restriction base="string">
     <pattern value="P[0-9]{2}DT[1-2]?[0-9]H[1-5]?[0-9]M[1-5]?[0-9]S"/>
  </restriction>
</onElement>
```

Fig. 2. Examples of service assumptions

It is worth noticing that we focus here on data changes (e.g. format, range, etc.). Changes regarding the structure of the service responses can be easily detected, because they require the interface definition of the service, written for instance in the Web Service Definition Language (WSDL) [3], to be changed. Adopting a new service interface involves rebinding and recompiling, and the compiler is able to detect syntactic incompatibilities. What the compiler cannot detect is (subtle) semantic incompatibilities (such as the change of a list item separator). This requires regression (audit) testing and test prioritization.

In our approach, service integrators specify their service assumptions explicitly, in order to simplify and automate audit testing of integrated services. For the specification of the integrator's assumptions, we propose an XML based assumption specification language. A service assumption consists of an XPath reference [5] to an element in the response under consideration and it can include data restrictions regarding that element. Data restrictions have the same format as those defined in the W3C XML Schema [4]. Service assumptions specify what a client expects from a service for its own purposes. Therefore, the data restrictions specified by one service integrator may differ from those in the service definition (e.g. in the WSDL interface of the service) or from those specified by another integrator. We refer the reader to our technical report [13] for more information regarding the structure and benefit of integrator's service assumption definition.

The listing in Figure 2 shows three examples of service assumptions. The first one, lines 1÷6, says that the client expects the length of the title to be between 5 and 256. The second one, lines 7÷13, specifies the possible values of *conditionId*. The last one, from line 14, constrains the *timeLeft* element to a regular expression. The restrictions defined in these examples can be used to validate the values of the elements of any XML response (e.g. the SOAP message) that matches the corresponding XPath expressions.

The Change Sensitivity Test Prioritization (CSTP) technology developed within the FITTEST project is aimed at prioritizing a set of test cases based on their sensitivity to external changes, e.g. changes at the service providers or at the parties that the system under test (SUT) interacts with [14]. The approach works based on the service assumptions of service integrators.

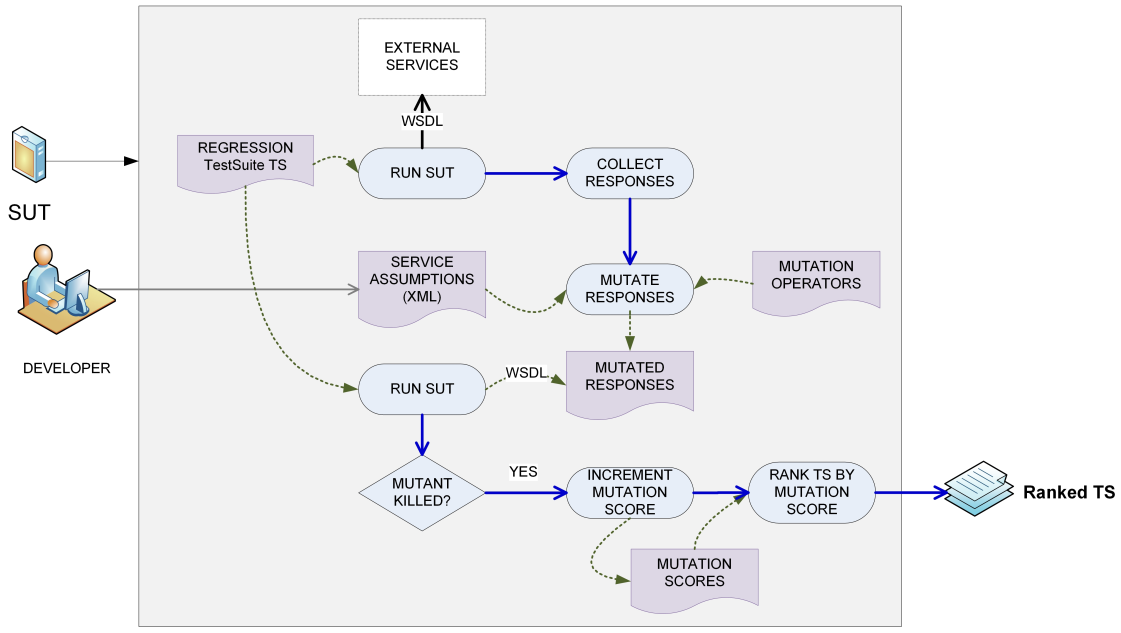

Fig. 3. Activities performed when applying CSTP

The measuring of change sensitivity is divided into six steps (see Figure 3): *(1)* executing the SUT on the regression test suite TS; *(2)* monitoring the service requests and collecting the response messages; *(3)* generating mutated responses by means of mutation operators, based on service assumptions; *(4)* for each test case, running the SUT and comparing the behaviour of the SUT when receiving the original response and when the mutated ones are received. Then, *(5)* the sensitivity (mutation) score is calculated and *(6)* test cases are ranked by mutation score.

3 Design of the case study

3.1 Context

The study was executed at IBM Research Lab in Haifa. More specifically within the research team responsible for building the testing environment for future developments of an IT Management Product (IMP) (similar to [2]), a resource management system in a networked environment (more details in Section 3.3). At IBM Research Lab, the developers conduct limited amount of testing;

the testing itself is conducted by a dedicated research team, whose work consists of testing the new versions of the IMP in a simulated environment (developed by this team) in which the system is executed. The testing activities described in this paper have been conducted in such simulated testing environment.

One of the objectives of the team is also to identify whether current regression testing practices could be improved or complemented by using some of the new techniques that have been introduced by the FITTEST EU project. For this purpose, the IBM Research team has used the FITTEST techniques and compared the results with the testing practices currently used.

Since the IMP is a mature system and in order to be able to measure the fault-finding capability of the introduced technology, several faults were injected into it (within the simulated environment) to mimic problems that can potentially be exhibited by such a system.

3.2 Objective - What to achieve?

IBM wanted to evaluate the Change Sensitivity Test Prioritization technique developed as part of the FITTEST project to find out whether it is applicable to the selected SUT and how well it compares to current regression testing practices at IBM. Following [19] and focusing on the three measures of applicability (or usability, as defined by ISO 9241-11), we consider: efficiency, effectiveness, and subjective satisfaction. Correspondingly, the research questions for the case study are the following:

***RQ1* (Effectiveness)** Can the test cases prioritized by the Change Sensitivity Test Prioritization technology contribute to the effectiveness of regression testing when it is used in the testing environments of IBM Research?

***RQ2* (Cost)** How much effort is required to introduce the Change Sensitivity Test Prioritization technology into the regression testing processes used at IBM Research? What cost saving does the technology support provide?

***RQ3* (Subjective satisfaction)** How satisfied are IBM testing practitioners during the usage of the technique, when applied in their real testing environment?

3.3 Objects

The objects used in this study are: the Software Under Test (SUT), the existing testing suite of IBM (TS_{ibm}), and the faults that were injected. Each of these are described below.

The SUT As stated earlier, the IBM IT Management Product is a distributed application for managing system resources in a networked environment. Its Management Server (MS) is an application that communicates with multiple managed clients and with users of the management system (typically system administrators). Managed clients are physical or virtual resources distributed over a network.

Mature versions of the IT Management Product system are used by IBM customers for managing IBM hardware and virtual devices, such as servers, Virtual Machines (VMs), switches and storage devices. The application has been developed for several years by IBM. IBM considers this application to be an important product, hence IBM is keen on assuring its quality through extensive testing. IBM Research has developed a simulated environment to allow for improved testing of future versions of the system. Consequently, this simulated environment is part of the object of our study.

The case study is to be performed on some new versions of this simulated system which are still under development and have not yet been released for customer use. However, the case study system shares data structures and protocols with the versions available to customers. As a result, public resources available for the production version are relevant to the case study as well.

The term end-point is used with reference to the managed resources on client nodes, from the point of view of the management server. The MS executes operations on the end-points either per request from the users, or autonomously based on defined policy (e.g. a recovery attempt after an end-point failure). The end-points have two-way communication with the MS. The communication operations supported are listed below (not all available for testing in this case study):

1. Discovery: MS finds what end-points are reachable;
2. Control: MS sends commands (configuration, OS restart) to end-points;
3. Query: MS queries status and configuration of end-points;
4. Report: End-points report problems to MS, and recovery from problems; and,
5. Deploy: Deploy and obtain disposal of end-points.

The MS keeps an inventory of the current managed resources and their states in a standard database. The case study system supports several protocols and message structures. In the case study, we focus only on HTTP messages. In the case study system, these are the easiest to capture, interpret, and modify for the case study purposes. In Figure 4 we present an example of two messages, a GET request and a response in JSON format.

```
2013:02:28:17:20:08;GET
https://.../ibm.com:8422/ibm/.../rest/IMP/workloads/500165838;;200;{
   "workload" : {
      "hosts" : {
         "uri" : "https://.../workloads/500165838/hosts"
      },
      "createBy" : "root", "resilient" : "false",
      "approvalRequired" : "false",
      "oid" : "500165838",
      "state" : {
         "label" : "Started",
         "id" : 8
      },
      "virtualServers" : {
         "uri" : "https://.../workloads/500165838/virtualServers"
      },
      "specificationVersion" : "0.0", "changedDate" : 1362064498,
      "metrics" : {
         "uri" : "https://.../Server/500165838/.../monitordate"
      },
```

Fig. 4. An example of two messages, a GET request and a response in JSON format.

In this message, at time 2013:02:28:17:20:08 there was a request to get the status of the workload with id 500165838. Upon successful execution, this service returns an HTTP status code of 200 (OK) and the response in JSON format containing relevant information about the workload. A full list of the possible HTTP requests in the API of IBM IT Management Product can be found in [2].

The existing Test Suite that was used for comparison with current practice (TS$_{ibm}$) Most of the testing effort is done by production teams outside of IBM Research. The IBM Research team has obtained a set of ten representative test cases from the external production teams. These form the IBM test suite (TS$_{ibm}$) to be prioritized for audit testing of the involved services.

The faults that were injected IBM Research has injected ten representative faults into the SUT for the evaluation of the different testing techniques. Those 10 injected faults are based on real faults that were identified in earlier versions of the product. They are listed in Table 1, together with their priority (based on the severity of the fault) that have been identified by domain experts at IBM research.

These faults have been manually injected into the code of the simulated system. In order to distinguish them from other failures, we added a distinctive comment in the log files generated by

Table 1. The faults injected into the SUT

Num	Injected Fault	Priority
F1	Fails to create unique new IDs	LOW
F2	Fails to open a JSON file	LOW
F3	Fails to validate workload size when resizing	MEDIUM
F4	Fails to delete workload fails	HIGH
F5	Fails to find a specific object ID	MEDIUM
F6	Fails to create workload when resources are not enough for a deployment	HIGH
F7	Fails to create a virtual server	HIGH
F8	Fails to create a virtual disk	HIGH
F9	Fails to get compatibilities	LOW
F10	Fails to write to a JSON file	MEDIUM

IBM Research's simulation environment. In our case study we use these injected faults as a measurement to evaluate the fault-finding capability of the FITTEST tool for test case prioritization.

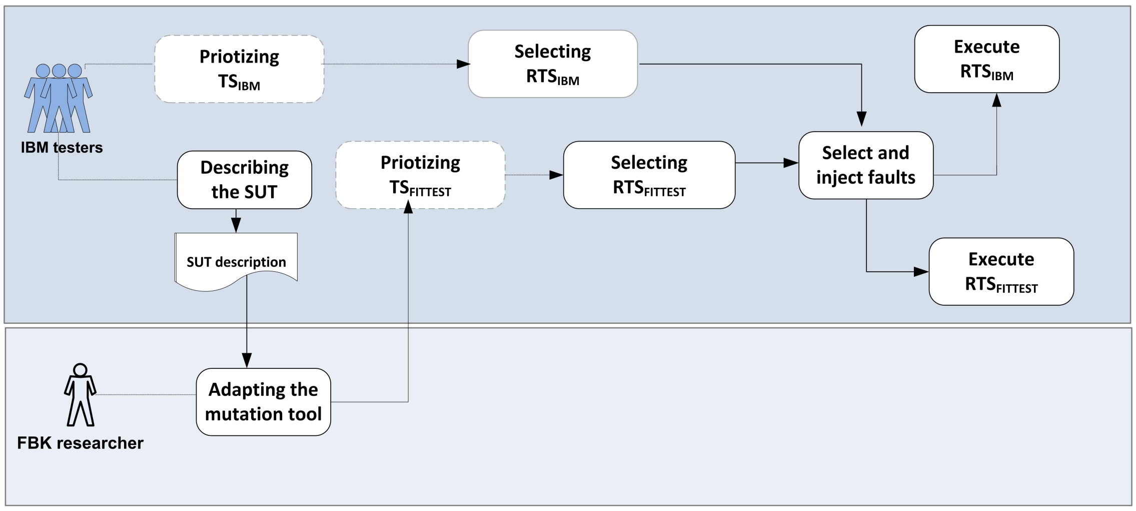

Fig. 5. The steps defining the protocol of the study

3.4 Cases - What is begin studied?

Current regression testing prioritization techniques used at IBM The prioritization of the tests selected for regression testing is done by the IBM Research team with the goal of maximum fault detection rate at the minimum possible time. This prioritization activity was done manually and substantial domain knowledge was needed to make decisions about the best order in which to execute the test cases. For the IBM Research domain experts, the tests with higher priority are those that cover most of the key scenarios of the SUT. The order chosen by IBM is: $T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9$ and T_{10} (in the following, this order is indicated as RTS_{ibm}).

When regression testing time is limited, the IBM Research team selects the first 4 test cases from RTS_{ibm} to be executed in the specified order (i.e. T_1, T_2, T_3, T_4), to allow for fast regression testing.

The FITTEST regression testing prioritization techniques This technique (see Section 2) comes with a tool that generates mutated messages, runs test cases, and ranks them based on their change sensitivity automatically. The test prioritization technique consists of the following key tasks:

1. Monitoring the original messages (requests, responses) exchanged between the SUT and the "external party" (either an external service provider or another node of the system). Test cases are executed against the SUT and message exchange between the two sides is monitored and used to generate mutated messages.
2. After obtaining the responses from the external party, the test set is re-executed with mutated responses.
3. Test cases that detect the mutated responses are considered sensitive to changes, and are ranked higher than others.

When a service provider changes or is replaced by another service provider, the test cases ordered by change sensitivity will be used in regression testing; those that are more sensitive to changes will be executed first. This helps developers detect faults early and find more faults within a limited amount of testing time.

In our case study, the SUT is the IBM system, where we consider the management node as our SUT, while the end-point nodes that the management node controls through their services are external services for the SUT. Communication between the SUT and the end-point services, based on JSON messages, is monitored and used to generate mutations and rank the test cases. The order produced by the FITTEST tool is indicated as RTS$_{fittest}$.

3.5 Subjects - Who apply the techniques?

The subjects are employees of IBM Research and a reseracher from Fondazione Bruno Kessler (FBK), a research center located in Trento, Italy[8], both members of the FITTEST consortium.

The IBM Research subject is a 30-year old senior tester from IBM Research with 10 years of software development experience, 5 years of experience with testing of which 4 years are of domain knowledge related to a system similar to the SUT. The tester had no previous knowledge of the FITTEST tool and holds a Computer Science degree.

The FBK subject was a researcher with more than 10 years of software development experience and more than 5 years of research experience in software testing and analysis.

3.6 Variables - What is being measured?

The key *independent variable* of the study is the Regression Test Suite (RTS) produced by each prioritization technique being assessed (i.e., RTS_{ibm} vs. $RTS_{fittest}$). Other factors to consider are the complexity of the industrial system and the level of experience of testers at IBM Research who use the FITTEST technique.

The *dependent variables* measure the applicability of the FITTEST prioritization technique in terms of effectiveness, efficiency (costs) and subjective user satisfaction. The respective metrics definitions are:

1. Effectiveness:
   (a) APFD$_C$ (Cost-cognizant Average Percentage of Faults Detected) [8]: Area below the weighted percentage of detected faults vs. test suite fraction curve, where faults are given a weight proportional to their severity (HIGH = 4; MEDIUM = 2; LOW = 1).
   (b) AP$_H$, AP$_M$, AP$_L$ (Average Position) [18]: average position of the first test case that detects each high, medium and low severity fault.

[8] URL: http://www.fbk.eu

2. Costs:
   (a) Technology transfer and training effort (man-hours) needed to apply the FITTEST technology to the IBM case study.
   (b) Human effort needed to create the manual test suite order.
3. Subjective satisfaction:
   (a) Interviewing the IBM subjects about satisfaction and perceived usefulness.

3.7 How the case study was executed

The case study protocol is depicted in Figure 5. In order to respect IBM business policies and enable the subjects to collaborate and measure the data identified in the previous section, the following steps were taken:

1. Select test suite TS for this case study [performed by the IBM Research subject].
2. Send SUT specification to FBK researcher [performed by the IBM Research subject].
3. Adapt the FITTEST tool to IBM specific SUT [performed by the FBK subject].
4. Execute IBM prioritization techniques to obtain RTS_{ibm} [performed by the IBM Research subject].
5. Execute FITTEST prioritization techniques to obtain $RTS_{fittest}$ [performed by the IBM Research subject].
6. Select the first 4 test cases (see Section 3.2) [performed by the IBM Research subject].
7. Select and inject faults [performed by the IBM Research subject].
8. Execute RTS_{ibm} and $RTS_{fittest}$ [performed by the IBM Research subject].

4 Results

Using the automated change sensitivity test prioritization technique on the SUT we obtained the following ranking $RTS_{fittest}$, shown above the RTS_{ibm} ranking:

RTS_{FITTEST}: T6 T8 T9 T10 T2 T1 T3 T4 T5 T7

RTS_{IBM}: T1 T2 T3 T4 T5 T6 T7 T8 T9 T10

RTS_{ibm} is the natural order from T1 to T10 while in $RTS_{fittest}$, T6 comes first, then T8, T9, T10, and so on.

Table 2 shows the fault detection matrix of the 10 test cases versus the 10 faults. As we can see, 3 faults (F2, F5, F10) are detected by all test cases, while 4 others (F3, F4, F7, F8) are not detected at all.

Fig. 6 shows the cumulative sum of the fault weights for the faults that are detected by each test case in the horizontal axis, which means T1, T2, T3, T4, etc. for RTS_{ibm} and T6, T8, T9, T10, etc. for $RTS_{fittest}$. T1 (the first test case of RTS_{ibm}) exposes all faults detected by the prioritized test suite, i.e., F1, F2, F5, F6, F9, F10, for a total weight of 11. T6 (the first test case of $RTS_{fittest}$) exposes the same faults, with one exception, fault F9. Correspondingly, in Fig. 6 the cumulative weight for FITTEST at position 1 is 10 (in fact, F9 is a low severity fault weighted 1). The next test case in $RTS_{fittest}$, T8, does not expose any fault yet to be detected. The next test case, T9, reveals F9, bringing the histogram at position 3 to the maximum value, i.e., 11.

Based on the histograms in Fig. 6 it is possible to compute the APFD$_C$ metrics, whose values are shown in Table 3. In the same table, the values for the average position metrics, divided by fault severity, are shown. In terms of APFD$_C$ metrics the two rankings $RTS_{fittest}$ and RTS_{ibm} are very similar to each other. This is also quite apparent from the histograms in Fig. 6, where the only difference is due to a low severity fault, F9, which is detected by the third test case in $RTS_{fittest}$, while it is detected immediately by RTS_{ibm}.

Table 2. Fault detection matrix of the test cases. "1" means that the corresponding fault is detected, "0" means the otherwise. Rows S, W show severity and associated weight.

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
S	L	L	M	H	M	H	H	H	L	M
W	1	1	2	4	2	4	4	4	1	2
T1	1	1	0	0	1	1	0	0	1	1
T2	1	1	0	0	1	1	0	0	1	1
T3	0	1	0	0	1	1	0	0	0	1
T4	0	1	0	0	1	0	0	0	0	1
T5	0	1	0	0	1	1	0	0	0	1
T6	1	1	0	0	1	1	0	0	0	1
T7	1	1	0	0	1	1	0	0	0	1
T8	1	1	0	0	1	1	0	0	0	1
T9	1	1	0	0	1	1	0	0	1	1
T10	1	1	0	0	1	1	0	0	0	1

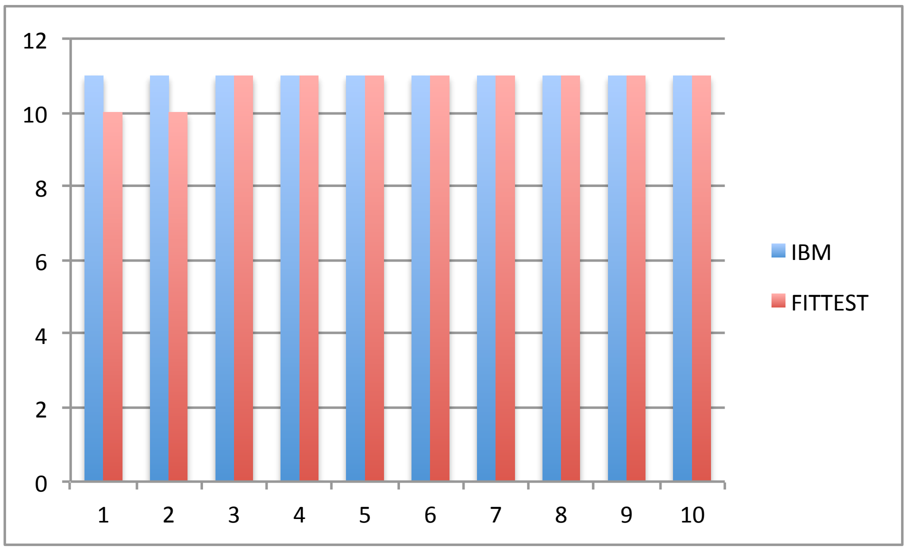

Fig. 6. Weighted fault detection vs. ordered test cases

Table 3. Effectiveness metrics

	APFD$_C$	AP$_H$	AP$_M$	AP$_L$
RTS_{ibm}	0.418	1	1	1
$RTS_{fittest}$	0.41	1	1	1.6

The average position of the first fault revealing test case confirm the observations made by looking at the APFD$_C$ metrics. There is no difference in the average position of the fault revealing test cases for high and medium severity faults, while for low severity faults the average fault revealing position is slightly higher for $RTS_{fittest}$ (see Table 3).

The IBM team considers a budget of 4 test cases as the lowest testing budget to be used when a fast regression cycle has to be executed. With such testing budget the two rankings ($RTS_{fittest}$ and RTS_{ibm}) are equivalent in terms of total fault detection capability. RTS_{ibm} is slightly preferable because it detects a low severity bug slightly earlier than $RTS_{fittest}$.

***RQ1* (Effectiveness)** *Test suites prioritized automatically by means of the FITTEST Change Sensitivity Test Prioritization technology are comparably effective as the test suites prioritized manually by expert IBM professionals.*

We analyzed the effort needed to complete the experiment. Most of the time was reserved for technology transfer. The FBK researcher involved in the experiment had to understand the SUT (available as a black box) and had to adapt the FITTEST tool to the SUT's technology. The FBK researcher spent also some time to define the service assumptions used by the FITTEST tool. Since the FBK researcher had no prior knowledge about the SUT, the total amount of time devoted to technology transfer and tool setup is substantial (around 2 working days). However, it should be noticed that this cost is paid once for all and all test case prioritizations are obtained in a completely automated way after the initial technology transfer and tool setup effort.

The ranking produced by IBM was obtained in a short time (the order of a few minutes) by an IBM professional with substantial domain knowledge and previous experience. However, it should be noticed that the manual prioritization effort tends to increase more than linearly, because of the cognitive difficulties faced by the human brain when a large number of items needs to be compared. For larger test suites the manual test case prioritization effort would increase substantially, making the cost saving greater than the one observed in our experiment (a few minutes of an expert developer).

It should be also noticed that the domain knowledge and experience required for an effective and efficient ranking of the test cases might be unavailable. In such a context, the FITTEST solution provides a comparable ranking at zero extra cost (excluding the initial technology transfer and tool setup costs). When the number of test cases becomes large and unmanageable by humans and when the developers involved in the regression testing activities have limited domain knowledge and experience, the FITTEST automated test case prioritization technology can provide substantial benefits and cost savings.

***RQ2* (Costs)** *The technology transfer and tool setup costs can be estimated in a few working days. The cost savings are small if experienced developers are involved and the test suite to be prioritized has a small size; we expect substantial savings and benefits when less experienced developers are required to prioritize larger test suites.*

In terms of subjective satisfaction, the testing technique itself was found rather easy to understand and simple to use. However, the testers have indicated some difficulties that made the testing in the real environment cumbersome and time-consuming. Specifically, because the FITTEST technique works on output logs in a specific format, and since the real system does not generate that exact format, there was a need for a manual, time consuming change either of the output logs or of the source code generating them. The testers suggested that, had there been an automated translator for this, it should have made the work with the FITTEST technology much simpler and faster. Of course, this had no effect on the testing technology itself, which proved efficient, effective and easy to use (once the log translation was handled).

***RQ3* (Subjective satisfaction)** *Despite having a technical issue with the differences in log format, the developers found the technique effective and easy to use.*

4.1 Threats to validity

The main threats that could affect the validity of this case study are:

Threats to *construct validity*, concerning the relation between theory and experimentation: To address these threats we adopted well-known and widely used metrics (APFD, AP) to assess the effectiveness of the prioritization technique. For costs, we measured the actual effort in person-hours. For the subjective evaluation, we used structured interviews.

Threats to *internal validity*, concerning factors that could influence our results: the setup of the FITTEST tool was made by the tool author, an FBK researcher, to ensure optimal configuration. Results may be different if the tool is not configured and used properly.

Threats to *conclusion validity*, concerning the relationship between the treatment and the outcome: we have drawn our conclusions based on objective measurements whenever possible. Some extrapolations, not supported by the experimental data (e.g., the tool behavior for large test suites) should be subjected to empirical validation.

Threats to *external validity*, concerning the generalization of our findings: we conducted a single case study, so generalization to other cases should be done with care. We expect similar results in contexts comparable to the IBM one.

5 Conclusions and Future work

We have presented a "which is better" [11] case study for evaluating FITTEST regression testing tools with a real user and real tasks within a realistic industrial environment of IBM Research. The design of the case study has been done according to the methodological framework for defining case studies presented in [19]. Although a one-subject case study will never provide general conclusions with statistical significance, the obtained results can be generalized to other similar software in similar testing environments of IBM Research [20, 10]. Moreover, the study was very useful for technology transfer purposes: some remarks during the study indicate that the FITTEST techniques would not have been evaluated in so much depth if it would not have been backed up by our case study design. Finally, having only limited number of subjects available, this study took several weeks to complete and hence we overcame the problem of getting too much information too late.

The objective of this research was to examine the advancements of the FITTEST automated regression testing tools and validate their potential to improve current testing practices at IBM Research. The results of this case study are:

- On the IBM case study, the test suite automatically prioritized by the FITTEST tool resulted to be almost equally effective as the test suite prioritized manually by IBM experts. The advantage of automated test suite prioritization is that it does not require the involvement of any expert developer. Moreover, for large test suites the manual approach becomes quickly impractical, while the automated one scales well to orders of magnitude above the test suite size considered in this study.
- Introducing a new technology in an industrial setting is known to require some effort and time. In our experiment, this was deemed acceptable by the IBM team. The cost savings observed in our experiment were marginal because of the small test suite size. We might be able to extrapolate larger savings when the test suite size increases.
- The subjective satisfaction of the tool users at IBM was definitely high. Users appreciated the automation and objectivity offered by the FITTEST technology. Any complex human decision making, which is required when the manual approach is used, is ruled out by the FITTEST tool, which determines the test case ranking based on objective sensitivity measures.

In summary, the FITTEST test case prioritization technology produces effective rankings with no need for expert knowledge and human decision-making. Even less skilled testing teams could benefit of high quality test case prioritization thanks to this technology. On large test suites we expect also to achieve major cost savings.

Moreover, from the FITTEST project's point of view we have shown that the FITTEST tools are useful within the context of a real industrial case and scale up to automate the test case prioritization for regression testing within a real industrial context.

For future work, we are planning to compare with other test case prioritization schemes, since it could be that effectiveness metrics for RTS_{ibm} and $\text{RTS}_{fittest}$ are just comparable and both ineffective. Moreover, we need to test a larger category of test suites for an industrial system to get a more accurate picture.

Acknowledgment

This work was financed by the FITTEST project, ICT-2009.1.2 no 257574.

References

1. Simple object access protocol (soap). http://www.w3.org/standards/techs/soap. Accessed October 2013.
2. Vmcontrol system. http://pic.dhe.ibm.com/infocenter/director/pubs.
3. Web service description language (wsdl). http://www.w3.org/TR/wsdl20. Accessed December 2010.
4. Xml schema. http://www.w3.org/XML/Schema. Accessed December 2010.
5. Xml path language (xpath). http://www.w3.org/TR/xpath/, November 1999. Accessed December 2010.
6. K. Channabasavaiah, K. Holley, and E. Tuggle. Migrating to a service-oriented architecture. *IBM DeveloperWorks*, 16, 2003.
7. T. E.J.Vos, P. Tonella, J. Wegener, M. Harman, W. Prasetya, and S. Ur. Testing of future internet applications running in the cloud. In S. Tilley and T. Parveen, editors, *Software Testing in the Cloud: Perspectives on an Emerging Discipline*, pages 305–321. 2013.
8. S. G. Elbaum, A. G. Malishevsky, and G. Rothermel. Incorporating varying test costs and fault severities into test case prioritization. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE)*, pages 329–338, 2001.
9. R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, University of California, Ivrine, 2000. Chapter 5 - Representational State Transfer (REST).
10. W. Harrison. Editorial (N=1: an alternative for software engineering research). *Empirical Software Engineering*, 2(1):7–10, 1997.
11. B. Kitchenham, L. Pickard, and S. Pfleeger. Case studies for method and tool evaluation. *Software, IEEE*, 12(4):52 –62, July 1995.
12. H. Ludwig, A. Keller, A. Dan, R. King, and R. Franck. A service level agreement language for dynamic electronic services. *Electronic Commerce Research*, 3:43–59, 2003. 10.1023/A:1021525310424.
13. C. D. Nguyen, A. Marchetto, and P. Tonella. A simple format for the specification of service integrator assumptions. Technical report, Fondazione Bruno Kessler, 2010.
14. D. C. Nguyen, A. Marchetto, and P. Tonella. Change sensitivity based prioritization for audit testing of webservice compositions. In *Fourth International IEEE Conference on Software Testing, Verification and Validation, ICST 2012, Berlin, Germany, 21-25 March, 2011, Workshop Proceedings*, pages 357–365, 2011.
15. L. Richardson and S. Ruby. *RESTful web services.* O'Reilly, 2008.
16. G. Rothermel and M. J. Harrold. A safe, efficient regression test selection technique. *ACM Transactions Software Engineering Methodology*, 6(2):173–210, Apr. 1997.
17. G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Prioritizing test cases for regression testing. *IEEE Transactions on Software Engineering*, 27:929–948, 2001.
18. L. H. Tahat, B. Korel, M. Harman, and H. Ural. Regression test suite prioritization using system models. *Software Testing, Verification and Reliability*, 22(7):481–506, 2012.
19. T. E. J. Vos, B. Marín, M. J. Escalona, and A. Marchetto. A methodological framework for evaluating software testing techniques and tools. In *12th International Conference on Quality Software, Xi'an, China, August 27-29*, pages 230–239, 2012.
20. A. Zendler, E. Horn, H. Schwartzel, and E. Plidereder. Demonstrating the usage of single-case designs in experimental software engineering. *Information and Software Technology*, 43(12):681 – 691, 2001.