Regression Testing as a Service*

[1]Sheng Huang, [2]Zhong Jie Li, [2]Ying Liu, [3]Jun Zhu, [1]Yang Hua Xiao, [1]Wei Wang
[1]Fudan University, China [2]IBM China Research Lab [3]Peking University, China
{shhuang, shawyh, weiwang1}@fudan.edu.cn {lizhongj, aliceliu}@cn.ibm.com zhujun07@sei.pku.com.cn

***Abstract*—Selective regression testing involves retesting of software systems with a subset of the test suites to verify that modifications have not adversely impacted existing functions. Although this problem has been heavily researched, it has never been discussed in the context of SaaS (Software as a service). This paper presents the specific requirements, challenges and benefits in delivering regression test selection as a service (RTaaS). We will introduce how to design and implement a RTaaS platform. An implementation of RTaaS has been piloted and improved via several real projects in China market. The real customer cases illustrate that RTaaS is a cost-effective and easy way for software project teams to leap over technical barriers and tap into advanced regression testing selection technologies.**

Keywords-component; Regression Testing, SaaS

I. INTRODUCTION

Regression testing is the process of validating the modified software to provide confidence that the changed parts of the software behave as intended and that the unchanged parts of the software have not been adversely affected by modifications [1]. It is essential to software quality throughout its lifecycle. In a typical scenario, program P has been tested by a test suite T and then released. During maintenance, the developer D updates P to P' by adding new features or fixing defects. Regression testing over P' is required before it can be released.

Nowadays the most common regression testing strategy in industry is to rerun all the test cases or select partial test cases based on testers' judgments. Some big software companies may choose to rerun the entire test suite to guarantee the coverage of changes, especially when the test cases are automatic. However, for most companies, *rerun all* are unaffordable due to the large numbers of test cases. This is also true for small and medium independent software vendors (ISVs). A survey over a couple of ISVs in China market showed that most of them only rerun a small portion of test cases selected based on testers' experience or simply skip regression testing. Many severe defects escape to production. This can be fatal to their business.

In the past several years, we developed a regression selection solution called ORTS [17] to automatically associate test cases with underlying software artifacts and support optimized regression test selection based on code change identification and analysis. We have deployed ORTS to IBM and several ISVs in China and proved its efficiency. However, we also have found that a lot of ISVs cannot afford to deploy a system to handle regression testing problems independently, including software and hardware investment and management efforts.

SaaS is software that is deployed over Internet and/or is deployed to run behind a firewall in your local area network. With SaaS, a provider licenses an application to customers as a service on demand, through a subscription or a "pay-as-you-go" model. SaaS is also called "software on demand". Now, it has become prevalent in many business tasks, including computerized billing, invoicing, human resource management, and service desk management. Some leading software companies also start to deliver the development and test tool software in service model. For example, IBM Rational recently announced a product called Software Delivery Services for Cloud Computing [19] to help the clients reduce capital and licensing expenses, decrease operation and labor costs, leverage IT infrastructure efficiently and shorten test provision cycle time.

Combining the essential difficulties of regression testing and the advantages of SaaS model, we proposed to leverage the SaaS business model to deliver regression testing service through web, called Regression Testing as a Service (RTaaS). We conducted a practical survey about the acceptance of RTaaS among ISVs. Our survey shows that these ISVs feel it imperative to own regression testing selection capability in a cheaper way, such as pay-as-you-go or subscription model. They expect that such a service could stop the quality degradation caused by poor regression testing and cut down the cost of their real software maintenance. In other words, the basic requirements for regression test selection tool include:

1) Scalable to large project size
2) Support different programming languages
3) Guaranteed change coverage

According to our survey and experience in providing such services, additional specific requirements for regression testing as a service (RTaaS) are summarized as below:

4) Intellectual Property is protected
5) No requirement for overmany prerequisite inputs
6) Easy to consume through web

Where, intellectual property protection is one of critical concerns of ISVs. Although a variety of academic regression test selection (RTS) techniques [1-6] have been proposed to select a cost-minimized subset to verify the modified

* We thank the members from IBM CDL for their active trail of RTaaS service and their comments for improving the RTaaS. We also thank the testers from QingFeng and Delver for their real usage experience feedbacks and suggestions as good candidates for future works. The paper was partially supported by NFSC under grants No.61003001; SRFDPHE with No.20100071120032. Correspondence author: Yanghua Xiao (shawyh@fudan.edu.cn).

978-0-7695-4463-2/11 $26.00 © 2011 IEEE
DOI 10.1109/ICWS.2011.39

program, most of these test selection approaches are based on source code. It can not solve the problem that no ISVs want to share the source code with SaaS platform. In addition, most of the source code based approaches have scalability problem. For a large commercial system, it is hard to build the data flow or control flow which is required in these approaches, although such approaches may work in Unit Testing. Chen [9] describes a specification-based method that doesn't need source code. However, the activity diagram is a prerequisite that doesn't usually exist in a software project.

In this paper we present a RTaaS implementation that proves to be able to meet these requirements and overcome mentioned challenges. This paper is organized as follows. A brief overview about ORTS is given in Section 2. Section 3 introduces the RTaaS usage scenario. Section 4 describes the service implementation details. Section 5 reports the case studies. Related works come in Section 6. Section 7 concludes the paper with future work prediction.

II. BRIEF OF ORTS TOOL

A variety of strategies [8] have been proposed to select a subset of regression tests for execution to verify the modified program in academic research. However, most of these test selection approaches are source code based. For a large commercial system, it is hard to build the data flow or control flow required in these approaches [7]. In addition, only considering the changes in programming language level is not enough in large commercial systems. To the best of our knowledge, ORTS is the first tool that targets at regression testing selection for commercial Java applications. ORTS pursues the following three phases.

Phase 1: Scalable runtime profiling. We have done a survey on current commercial Java applications, and it shows that most of them are Web applications. Existing regression tools for Java applications such as Contest [18] lack of support to popular IT artifacts used in Java Web Applications such as JavaScript (JS), JSP. ORTS has the capability of capturing Java method invocations, JavaScript method invocations and JSP loading events. In this manner, ORTS guarantees that no IT artifact traversed during the execution is missed. In addition, the run time profiling incurs little overhead to test execution.

Phase 2: Build oriented change identification. Current testing services are likely to be distributed in different regions as the testing team is separated. The testing team always has no access to source codes. An alternative solution by decompiling binary files [10] is proposed, but it is illegal. In ORTS the change points are derived by analyzing the builds (EAR/WAR/JAR) of two versions. Besides the logic changes caused by modifications to Java/JS/JSP, the configuration level changes, are also taken into consideration. This feature gives ORTS the ability of identifying complete change points of commercial Java applications.

Phase 3: Optimized regression test suite. Similar to the safe regression strategy in [8], only test cases traversing change points would be selected out in ORTS, consequently lots of unnecessary test cases would be avoided. Even in this way, the test suite size may still exceed the deliver pressure. To address this problem, ORTS prioritizes the regression test suite in terms of risk and guides the rerun schedule under the time pressure.

Based on the three phase regression test selection strategy, we realized ORTS， a tool facilitating testers to generate optimized regression test suite for commercial Java applications. The whole design strategy is lightweight, making the regression test selection process more automated and effective, and scalable to commercial regression testing scenarios with resource and time constraint.

III. REGRESSION TESTING SERVICE SCENARIO

However, this tool requires each ISV to deploy ORTS to handle regression testing problems independently, including software and hardware investment and management efforts which are not expected by the ISV. Hence, a more convenient, cheap way for ISV to own the regression ability—RTaaS (regression testing as a service) is expected to be built.

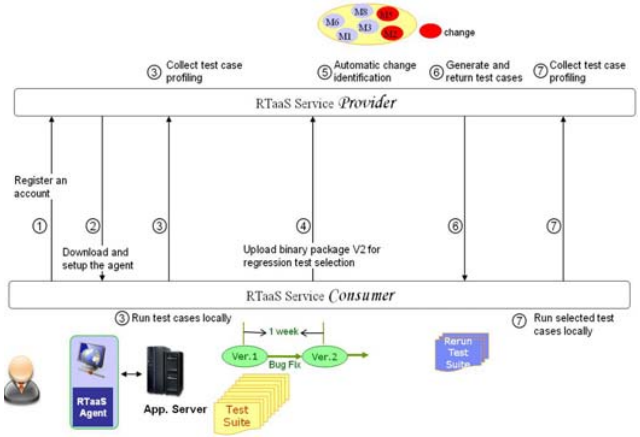


Figure 1 One RTaaS usage scenario

Fig. 1 shows one usage scenario of RTaaS based on ORTS technologies. In this scenario the RTaaS service is deployed in the Service Provider side, and the RTaaS Service Consumer, say the ISV, consumes the service through web based channel. This is a seven-step process. In step 1, the ISV visits the website and registers a RTaaS account. In step 2, the ISV downloads a RTaaS agent from the same website and setup it in local workstations. This agent will be used to capture the behavior of test case execution. From the behavior that records the invocation chain during test case execution, we build the mapping between test cases and software components – which components are covered by a test case. In step 3, the ISV runs the test cases for the initial application release (say version 1). The agent will collect the runtime behavior information (test case profiling) and send it back to RTaaS server. After all the test cases are executed some bugs are reported. One week later all the bugs are fixed and a new build is generated for regression testing. In step 4, the ISV uploads the binary package of version 2 to the RTaaS service. In step 5, the RTaaS service identifies the changed

components automatically. Then in step 6, the RTaaS selects the test cases covering any of these changed components and returns the list to the ISV. In step 7, the ISV gets the list and rerun the selected test cases. The test case execution behavior will be collected by the agent and sent back to the server. This completes a regression cycle. If there are further bugs or updates to the application, new builds will be uploaded and steps 4~7 will be repeated. We could find that with such kind of process, it is easy to build the linkage between test cases runtime trace and specific build under test in RTaaS Server side as the latest updated trace will be annotated as the trace executed over latest updated build.

IV. RTAAS SERVICE IMPLEMENTATION

As described in the introduction section, there are six major requirements and challenges for delivering regression testing as a service. They are key elements to a successful RTaaS. The three items at the top of Fig. 2 are primarily regression testing related, and the three items at the bottom are primarily related to general SaaS (but also related to regression testing, although not so tight).

This section will introduce the implementation architecture and then explain how the six success keys are supported.

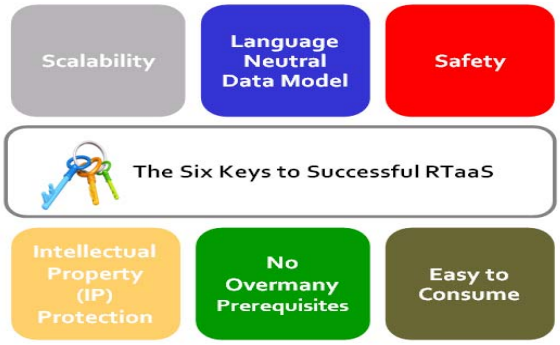


Figure 2 Six keys to a successful RTaaS

A. Architecture

Fig. 3 shows the architecture of our RTaaS implementation. Its server can handle multiple users concurrently. The target applications under test do not need to be installed on the RTaaS server; they can be deployed in the users' local environment, where an agent does need to be installed to do runtime profiling of the target applications during execution and to communicate with the RTaaS server.

On the agent side, the Weaver helps to do static instrumentation on the target applications. AspectJ [16] is used to instrument binary Java code, and a customized Ajax-like way is used to instrument JSP and JS code so that JSP loading events, form submission events and JS method invocations can be captured. Our experiments have proven that this approach is very lightweight in terms of the instrumentation time cost and profiling overhead. During test execution, the profiler will get the relevant events.

On the server side, users can upload the build to be under test of target applications. The first build is updated during project initialization. Once a new build is created by

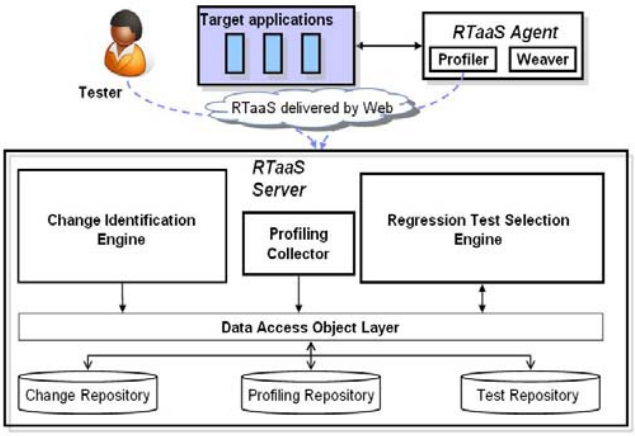


Figure 3 RTaaS system architecture

development team for regression testing, the build comparison is performed by the change identification engine, which extracts programming artifacts from the builds and identifies Java method changes by scanning the binary Java classes directly. It also runs a syntax analysis to identify JS method-level changes, JSP changes and configuration dispatch nodes changes. The profiling collector on the server side communicates with the profiler on the agent side to collect the profiling data and builds the linkage between test cases and backend events. Such kinds of linkage as a test runtime trace will also be marked with linkage to specific build run over. The regression test selection engine then uses the build-change and test-case traceability data to do its job, where the right test trace need to be analyzed with the build it run over. It returns all the test cases traversing change points. These test cases can be ranked by their risk score, which is calculated using factors like the number of change points covered, the change types, the invocation counts of the change points per test case, the bug history of test case.

There are different kinds of data involved (e.g., test cases, build changes, and profiling data) and they need good management. The data is managed in central repositories, facilitating access from the analysis components. That is why we do not allow the client to download/setup the whole application along with the agent in step 2 and use pay-per-use license associated with the application for simplicity. Otherwise, it will incur too much burden to service user.

B. Scalability

To satisfy the scalability requirement, we carefully considered the granularity of test case behavior monitored, i.e. the level test profiling occurs. As aforementioned, to track the end to end invocation chains during the test execution, code instrumentation is required.

Fine grained profiling occurs at branch or statement level. It can see which branches or statements are executed during the test case execution. With such information, the selection of impacted test cases due to changed branches or statements can be very precise. However, it induces too many interruptions and great system overhead to the application at runtime. Even for a moderately large J2EE application, the response time can become so bad that testing cannot go.

Therefore, the applicable area is mainly unit testing, not system level testing.

Coarse granularity profiling works at method level. It induces little system overhead to the original application, and the performance degradation is almost unnoticeable. About the preciseness, empirical study in [7] shows that at system level testing the coarse and fine level are similar in most cases.

This motivates us to choose the coarse granularity test profiling in RTaaS. Consequently, change identification will only need to capture method level elements and their changes, and the regression test selection algorithm can be simplified as well. These all contribute to improved performance compared to fine grained profiling, change identification and regression test selection.

C. *Language Neutral Data Model*

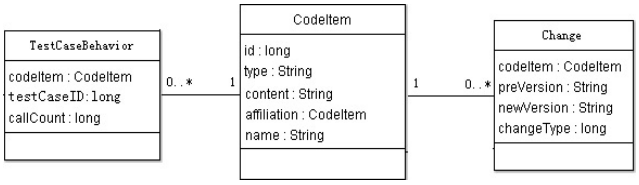


Figure 4 RTaaS data model (partial)

As Fig. 4 shows, the core regression testing data model is designed to be independent of programming language types. There is no specific data structure for any specific programming language. The test case, behavior, change model are all neutral and generic enough to handle any language types. The CodeItem could represent any programming artifact. New languages can be easily plugged in by implementing specific weavers, profilers and change identification engine adaptors. Now RTaaS supports J2EE and .Net applications.

D. *Safety for commercial application regression test selection*

A safe RTS technique [8] selects, under certain assumptions, all test cases (T') in the test T that may behave differently from P to P' by deriving all the test cases traversing the changed parts between P and P'. Safety is important because it ensures that T' reveals all of the regression defects in P'. As described earlier, customers' confidence on a RTaas service lies in its safety.

It is not trivial to find safe regression test suites. One of the difficulties is to identify all the changes and their impacts on the software behavior. We have surveyed current commercial Java applications and found that most of them are Java Platform Enterprise Edition (J2EE) applications. Besides Java, other programming languages such as JavaScript (JS) and JavaServer Pages (JSP) are popularly used. In addition, various frameworks are applied to make J2EE applications more scalable, maintainable, and structured. Model-View-Controller (MVC) is an example. Unlike pure Java applications, the logic of J2EE applications is controlled by the combination of code written by different programming languages and frameworks. Modifications to Java, JS, JSP, and configuration-level changes all can make P' behave differently.

The features described above create big challenges for existing RTS approaches because existing approaches handle only pure Java code while ignoring JS, JSP, and configuration files. Thus, existing approaches offer only poor safety.

To guarantee the safety, we should ensure 1) all behavior covering the change of interest is captured in the test profiling; 2) all concerned changes in different programming artifacts are identified for a new version; 3) the test traces linkage to specific build is distinguishable. Table 1 summarizes all the programming artifact units and related events that should be captured during test execution.

At runtime, all such events in a test-case execution will be monitored, captured and used to construct a behavior flow composed of nodes (artifacts) and transitions (execution events). When comparing two versions of the application to identify changes, any update that leads to behavior changes in test cases should be considered. Among others, the J2EE configuration files of them are also parsed and compared. Because of the syntax and semantics differences of different J2EE framework configurations, the generic behavior flow modeling, runtime profiling and change identification can be very complex – we call them hybrid test-case profiling and unified change identification. We'll report the details in another separate paper due to space limitation.

Table 1. Types of programming artifact units

Programming artifacts	Execution events
Java	Java method invocation
JSP	Script functions, JSP loading event, Form submit
JS	Script functions, JS method
Configuration files	Execution of dispatching nodes …

With such techniques, RTaaS has a guaranteed safety – a very competitive SLA. This means that customers can trust the complete coverage of regression defects after running the selected test cases suggested by RTaaS.

E. *Intellectual Property (IP) Protection*

RTaaS is designed to make customers comfortable in terms of IP protection. In response to customers' unwillingness to share source code and also reduce the code leakage risk at service provider side, RTaaS test instrumentation and change identification require only binary code. The on-premise agent can do instrumentation to the binary code directly, and the server side change identification can analyze the binary files of the application in a disassemble way. An extension of the standard JDK tool Javap [11] is utilized to accomplish this work as illustrated by Fig. 5. Disassemble is legal compared to decompilation, which is usually illegal. To our knowledge, a lot of regression test selection tools require the source code, and thus generally unqualified for being published as a service.

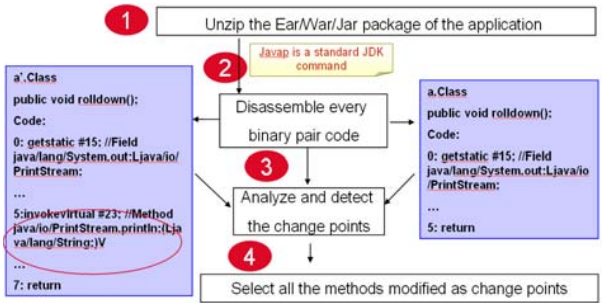


Figure 5 Change identification process

The change identification works as follows:

1) Unzip the Ear/War/Jar package of the application.

2) For each class pair in two versions, use Javap tool to disassemble them.

3) Analyze the assemble code in method pair to check if there are modifications.

4) Select all the methods modified as change points.

As an example, in a'.class method rolldown() a new functional call is added, thus we can identify that the method rolldown() is changed. All the changes leading to logic differences will be considered, including: *Method change*, including the change with the assignment of public member variables in class member methods. *Method deletion or addition*. For the test cases traversing the methods deleted or added, there must be an invocation entry deleted or added. Therefore, such kind of modification can be counted in. Notice that the implicit changes caused by method deletion or addition for Object-Oriented programming languages will also be considered by leveraging the method used in [6].

Renaming of member variables, addition/deletion/ modification and format change of comment lines that will not cause logic differences will be ignored. In this way, the number of change points is kept as small as possible, resulting in smaller regression set.

F. *No Overmany Prerequisites*

RTaaS only relies on two types of inputs: binary code and runtime profiles (linked to test case id). This is really attractive given the software development realities nowadays.

On one hand, many software projects don't have good requirement and design documentation, existing documents are poorly maintained so that they gradually lose synchronization with the code and become outdated. On the other hand, although model driven architecture existed for a long time, use of models is not common enough yet. And again, these models can be outdated too.

Even the documents and models are available and up-to-date, customers are reluctant to disclose them to external testing service providers. Note that here we're talking about online service providers that do business with massive SMBs (Small-and-medium businesses) who don't have the budget for normal testing outsourcing.

Under such constraints, we think that a regression testing service that requires no much prerequisite has a major advantage over those requiring software document or model specifications or source code.

G. *Easy to Consume*

As IBM champions, consumability is a description of customers' end-to-end experience with technology solutions. It's a higher-level concept than usability and includes aspects like "learning to use the product", "installing and configuring the product", "using and administering the product". To support good consumability, RTaaS has three design considerations: *Easy to learn*. There are also step-by-step tutorials on the website to enable self-study learning. *Easy to install.* At the customers' premise, only one agent is needed to install in their test environments. Customers don't need to care server side installation and update. *Easy to use.* Test manager and testers are the target user group. The only extra work for a test manager is to create projects and regression test case buckets, upload correct builds and sometimes profiling traces in the web based tool. The only extra work for a tester is to input a test case ID before executing it. No complex concepts. No need to understand internal analysis logic.

V. CASE STUDY

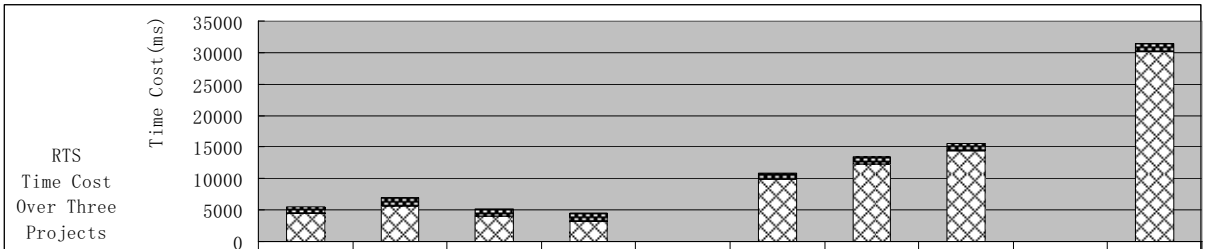
The implemented RTaaS has been deployed in a software park in China market. In this deal, the RTaaS is placed on a SaaS platform. There are also some other software engineering related services provided to ISVs in this software park. The software park aims at incubation of small ISVs in the software park by utilizing the services published on this platform. RTaaS has three real case experiences until now. Two of the consumers are the ISVs from this software park while the other is an internal IBM project team in IBM China Development Lab. The three case results are very positive with the evidence showing that RTaaS is convenient to consume through web as a service and is able to save the time and cost of regression testing dramatically together with the coverage assurance.

A. *Case background*

We collected the data over several releases of these projects. The first project A was an IBM internal project with 600 test cases, while project B with 50 test cases and project C with 194 test cases came from two ISVs in the software park: QingFeng and Delver.

Table 2: Project information

Project	#Files	KLOC	#Artifact Units	#Iteration	#TCs
A	1,642	1,960	14,220	1	600
B	162	38	1,227	4	50
C	472	185	2,705	3	194



	A2	A3	A4	A5		B2	B3	B4		C2
Test Selection	984	1328	1141	1360		860	1140	1188		1328
Change Identification	4537	5689	4032	3204		9922	12360	14391		30134

Figure 6. RTS time cost over three projects

Project A is stable enough through a long period of use, the main update was to fix bugs or perform minor function updates. We followed project B for one month of four iterations and project C for two months of three iterations. Both QingFeng and Delver are very small ISVs in the software park. They do not have formal testing team and always ignore regression testing in their past projects. They have been suffering from negative impact caused by changes.

Table 2 shows the project information including file number, KLOC, number of Artifact Units, number of iterations, and number of test cases (TCs). The projects vary by size: 38 KLOC, 185 KLOC, and 1960 KLOC for projects B, C, and A, respectively.

In these cases, we compared RTaaS with the *rerun all* strategy. Besides the regression test suites selected by RTaaS, the consumers also did us a favor to run the entire test suites for comparison.

B. Results related to regression testing specific requirements and challenges

First of all, RTaaS is a regression testing approach. There are primarily three regression testing specific requirements and challenges when delivered as a service: Scalability, IP Protection, and Safety.

- Scalability

Scalability is about whether RTaaS scales to large applications. The scalability oriented design and implementation approach has been described in section 3.2. To evaluate the actual scalability, we investigated the customers' testing experience and RTS time consumption. The testers told us that the testing went all the same as before when RTaaS is not used – there is no any performance degradation noticed. As the test case execution is a manual work, it doesn't make sense to collect actual execution time for comparison. The testers' experience is convincing and sufficient.

The RTS consists of three steps: package unzip, change identification, and test selection. In RTaaS, every time the consumer uploads a package, RTaaS runs the package-unzip process offline. As package unzip can be done offline, we can ignore the time it consumes. Thus, the RTS time here follows the formula: RTS time = change identification time + test selection time. As Orso proposed in [6], the scalability of RTS is important when applied to large applications. As Fig. 6 shows, each bar indicates the time cost of RTS time for each iteration (A1 indicates the 1st iteration for project A, etc) with RTaaS service. RTaaS finishes the change identification from 4 second to 32 second as the size of project increase from 38 KLOC to 1,960 KLOC. The test selection times are steady around 1~2 seconds. The performance demonstrated here is better than that of linear algorithms.

- Language neutral data model

The programming languages vary among different ISVs. It is not sharp to bind the RTaaS with one specific programming language. During our promotion activities of RTaaS in the software park, all the ISVs think that the service should support all the common languages. The results show that RTaaS could support their applications smoothly even when three programming languages are involved: Java, JavaScript and Java Server Page (JSP). Currently RTaaS could support Java, JavaScript, JSP with extendable interface in internal implementation as previous section describes. To add a new language, we only need to add a particular profiling and change identification component. In fact, the .Net programming extension has been ready for evaluation.

- Safety

Safety concerns the effectiveness of a selection strategy in discovering the regression defects. The effectiveness is measured by regression defects coverage rate: (discovered defects number by regression test suite / all regression defects number during the iteration). Although *rerun all* is absolutely safe, the high cost makes it unacceptable. We assume that the defects discovered in *rerun all* are just the total regression defects during iteration. In Fig. 7 the X axis indicates the iterations. The number besides the iteration code (A1, B1, ...) is the number of discovered defects. In all the iterations of the three projects, RTaaS selection could discover the same number of defects as rerun all. This observation is consistent with the theoretical analysis result. Three projects, eight iterations might not be significant numbers, but they're strong enough as these are very representative cases.

For project A, with RTaaS service the tester only need to rerun 25 test cases, which is quite small compared to 600 with rerun all. Assume each tester could run 10 test cases one day, RTaaS saves (600-25)/10 = 58 person days cost for this iteration.

Project B is a relative small application, and only has one tester. There are totally 40 test cases in B1 and 10 more test cases are added for the rest iterations as one new function is

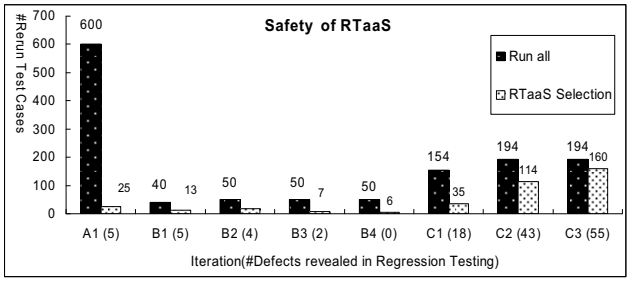


Figure 7 Safety of RTaaS

added. By leveraging RTaaS, the tester could finish the regression test all by herself in one or two days as she only need to run 13, 19, 7 and 6 test cases over iteration B1, B2, B3 and B4, respectively. The tester had been working under very high pressure in this project due to very short iteration cycles, even daily iteration. Without RTaaS, it will be crazy for her to rerun all the test cases to get enough confidence.

For project C, the ISV has three testers. After C1, there are some new components added into the application. Thus the total test case number increases from 154 to 194. We could find that the test case number selected by RTaaS is close to *rerun all*: C2 (194-114=80) and C3 (194-160=30). The reason is that this application is in its very early stage when big changes are happening and impacting most test cases.

From these results we could find that although the reduction rate varies for applications with different size, type, and maturity, RTaaS could always help the consumers reduce the time of regression testing with the safety confidence.

C. Results related to SaaS requirements and challenges

There are three requirements and challenges applicable to general SaaS: Language neutral data model, No overmany prerequisites, and Easy to consume. With that said, the context and content of the implementation is still regression testing specific.

- IP protection

No ISVs agree to share their source code with any third parties. But we are glad to find that all the ISVs in the case study agree to upload their binary packages to RTaaS for change identification. The owners of these projects take it for granted that they do not need to worry about IP protection issue by only providing binary packages together with some anti-recompile agreements signed with the RTaaS service provider—the software park platform operator.

- No overmany prerequisites

A lot of small ISVs do not have formal design documents, and it's unrealistic to ask them to provide activity diagrams, control flow graph, or data flow graph, etc. What's more, such small ISVs are fond of agile software process, where the face to face communication is valued higher than complex documents. As an example, the ISVs of Project B and C only have simple requirement description document. Here is a direct quote from one ISV: "We are always under heavy work pressure and do not have time to prepare documentations for regression testing. RTaaS is a good choice as we only need to provide the binary builds and then RTaaS will output the amazing regression test suite. The service also brings no extra burden to us in test case execution, as the only thing we need to do is to input the test case ID in the RTaaS agent."

- Easy to consume

To enable SaaS, the service should be easy to consume. It is ideal that the service can be learned by self-study instead of formal training.

In this case study, the consumers spend less than half day to master the basic steps to use RTaaS service by following the step-to-step guidance and try the default hello word sample after registration. The two ISVs in the software park claims that "It is easy to consume RTaaS as I only spend half a day to learn it. The only two mandatory steps are downloading the RTaaS agent and inputting the test case ID during test execution. It is meaningful because the half day we spent will save several days in future software iterations".

For project A, it is deployed behind the firewall. Although the RTaaS agent could not upload the test case behavior tracing data online due to the firewall limitation, RTaaS agent generates the local behavior tracing data copy in XML format. The tester could copy it and upload it to RTaaS server in a batch mode.

VI. RELATED WORKS

Software change is frequent and risky. Regression testing is necessary, but also costly. The retest-all strategy is straightforward and safe, but suffers from taking a lot of time and resources [8]. The selective regression testing strategy aims at finding a subset of test cases to rerun. Many techniques, such as program dependence graph [2], path analysis [3], dataflow [4], and graph walk [5], have been proposed for regression testing selection. All are source-code based. They can be applied effectively to regression testing at the unit or component level, but they are generally too expensive when applied to large programs. In addition, they are not IP protection friendly. The method proposed in [1] is safe for pure Java applications but not scaled to large applications. Existing code-based regression tools for large Java applications, such as DejaVOO [6], provide a safe approach to scaling regression testing for large Java applications, but it could handle only Java programming languages. In addition, it also requires the source code, which could be applied in RTaaS scenario. An alternative solution based on decompiling the binary files [10] of the builds can be leveraged, but it is not suitable for service delivered model. Chen [9] describes a specification-based method for regression test selection without the limitation of source code. However, the activity diagram is a prerequisite and activity-diagram-level changes are not accurate enough for practical commercial applications in RTaaS scenario.

A lot of researchers have been studying "Testing as a service" for several years. In [12], the author introduces a domain-specific MBT solution for graphical user interface testing of Symbian S60 smart phone applications. Yu [13] presents how to build a testing platform on web to support user acceptance testing by leveraging community resource. This topic mainly studies how to check the community

testing results through automatic audit mechanism. Some enterprises, such as HP [14] and IBM [15], have realized that delivering testing services through web or cloud platform is an industry trend and they have offered testing services through pay-as-you-go model. To our knowledge, all of existing solutions only address general testing problem, for example, how to build your testing excellence center on web and how to leverage the low cost testing resource on cloud platform. Neither industry solutions nor academic researches provide solutions to enable regression testing as a service presented in this paper.

VII. SUMMARY and Future Work

Selective regression testing involves retesting of software systems with a subset of the test suite to verify that modifications have not adversely impacted existing functions. SaaS is becoming an attractive software delivery model along with the prosperity of Cloud Computing. In this paper, we introduce how to combine the advantages of SaaS model and regression testing requirements to provide Regression Testing as a Service. We carefully analyze how it address the six keys of RTaaS success, including scalability, intellectual property protection, safety, language neutral data model, no overmany prerequisites, and easy to consume. This solution was implemented, piloted, and improved via several real projects in China market. The specific requirements, challenges and benefits in delivering regression test selection as a service were also summarized and shared. The case evaluation shows that RTaaS is an extremely cost-effective and easy way for software project teams to leap over technical barriers and tap into advanced regression testing selection technologies.

It is exciting that RTaaS is accepted by the consumers in the case study and the evidence shows that it really helps the consumers quite a lot in regression testing through web. We have also got some useful feedbacks which can be good motivations for future work.

It is expected to provide open interfaces to compose with other testing services or software engineering services. For example, a customer is using a certain test case management tool. They want existing test cases get imported into RTaaS directly, and RTaaS output the selected regression test cases to the test case management tool directly.

As RTaaS is able to link test cases to their runtime behavior, it is expected to provide additional services like test coverage report, which tells components covered and not covered.

Although RTaaS could reduce the test cases to rerun, it is still possible that testers don't have time to run all the selected test cases. This is a challenge to the safety of the RTaaS because the behavior history collected may be outdated for the test cases not rerun after some iteration. More work is needed to solve this problem.

REFERENCES

[1] M. J., Harrold, J. A. Jones, T. Li, D. Liang, A. Orso, M. Pennings, S. Sinha, S. A. Spoon and A. Gujarathi. Regression test selection for Java software. Proc. ACM Conf. on Object-Oriented Programming, Systems, Languages, and Applications (Tampa Bay, FL, 2001). ACM, New York, NY, 312-326.

[2] S. Bates and S. Horwitz. Incremental program testing using program dependence graphs. Proc. 20th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (Charleston, SC, 1993). ACM, New York, NY, 384-396.

[3] P. Benedusi, A. Cimitile, and U. D., Carlini. Post-maintenance testing based on path change analysis. Proc. Conf. on Software Maintenance (Scottsdale, AZ, 1988). IEEE Computer Society, Washington, DC, 352-361.

[4] M. J. Harrold and M. L. Soffa. An incremental data flow testing tool. Proc. 6th Intl. Conf. on Testing Computer Software (Washington, DC, May 1989). ACM, New York, NY.

[5] G. Rothermel and M. J. Harrold. A safe, efficient algorithm for regression test selection. Proc. Conf. on Software Maintenance (Montréal, Quebec, Canada, 1993). IEEE Computer Society, Washington, DC, 358–367.

[6] A. Orso, N. Shi and M. J. Harrold. Scaling regression testing to large software systems. Proc. of the 12th International ACM SIGSOFT Symposium on the Foundations of Software Engineering (Newport Beach, CA, 2004). ACM, New York, NY, 241-251.

[7] T. L. Graves, M. J. Harrold, J.M. Kim, A. Porter and G. Rothermel. An empirical study of regression test selection techniques. ACM Trans. Softw. Eng. Meth. 10, 2 (April 2001), 184-208.

[8] G. Rothermel and M. J. Harrold. Analyzing Regression Test Selection Techniques. IEEE Trans. Softw. Eng. 22, 8, (Aug. 1996), 529-551.

[9] Y. Chen, R. L. Probert and D. P. Sims. Specification-based regression test selection with risk analysis. Proc. Conf. of Centre for Advanced Studies on Collaborative Research (Toronto, Ontario, Canada, 2002). IBM Press.

[10] J. Zheng, B. Robinson , L. Williams and K. Smiley. Applying regression test selection for COTS-based applications. In Proceedings of the 28th IEEE International Conference on Software Engineering (Shanghai, China, 2006). ACM, New York, NY, 512-521.

[11] javap-The Java Class File Disassembler, Sun Microsystems. http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/javap.html

[12] A. Jääskeläinen, M. Katara, A. Kervinen, H. Heiskanen, M. Maunumaa, T. Pääkkönen. Model-Based Testing Service on the Web, Proceedings of the 20th IFIP TC 6/WG 6.1 international conference on Testing of Software and Communicating Systems: 8th International Workshop. Lecture Notes In Computer Science, Vol. 5047, 38-53.

[13] L. Yu, W. Zhao, X. F., Di, C. Kong, W.B. Zhao, Q.X. Wang, J. Zhu. Towards Call for Testing: An Application to User Acceptance Testing of Web Applications. 2009 33rd Annual IEEE International Computer Software and Applications Conference, 166-171.

[14] HP Testing-as-a-Service, https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-23%5E42007_4000_100__

[15] Smart Business Development and Test Cloud, http://www-935.ibm.com/services/us/index.wss/offering/midware/a1030965

[16] Eclipse AspectJ. http://www.eclipse.org/aspectj/.

[17] S. Huang, J. Zhu and Y. Ni. ORTS: a tool for optimized regression testing selection. OOPSLA Companion 2009: 803-80

[18] http://www.alphaworks.ibm.com/tech/contest

[19] www.ibm.com/software/rational/info/cloud-service