Volume 6, Issue 1, January 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Testing Autonomous System Integrations Utilizing Fuzzing Technique

S. Faizullah
Rutgers University, Piscataway, NJ,
USA

Abstract: ***With the emergence of specialized services and service providers, fueled by proliferation of IT services outsourcing, with some providers gaining leading positions in marketplace today, challenges are faced by teams who are tasked to deliver integration projects with much desired efficiencies both in cost and schedule as well as security. Such integrations are growing both in volume and complexity. Integrations between different autonomous systems such as workflow systems of the providers and their customers are an important element of this emerging paradigm. The interconnections utilizes XMLs via HTTPS POST, REST/RESTful, or SOAP. We have earlier presented an efficient model to implement such interfaces between autonomous workflow systems with close attention given to major phases of these projects, from requirement gathering/analysis, to configuration/coding, to validation/verification, several levels of testing and finally deployment. In this paper, we will provide details of the testing strategy for these autonomous system integrations domain software. In particular testing functionality and the security testing of web services independent of its customers/partners side technology, whereby for the latter we utilize fuzzing techniques. We are encouraged by the results both in projects in academic lab and as practitioners for real life projects.***

Key Words: ***Fuzzing techniques, SOA, software testing, security testing, software validation/verification, cloud computing, unit testing, integration testing, test oracles, empirical studies, regression testing.***

I. INTRODUCTION AND BACKGROUND

In today's globally connected world with the proliferation of cloud systems, large corporations and governmental bodies rely on IT Service Providers to manage their infrastructures, data centers, applications, workflow system, and sometimes their entire IT organization. Distributed nature of IT infrastructure management is emerging as complex and challenging service where it presents providers with unique opportunities and huge financial rewards if such management is executed efficiently. This model necessitates, based on service-oriented-architecture (SOA), among other activities, the need for connecting customers' and their partners' autonomous systems to those that Service Providers deploy to manage their infrastructure- such interconnectivity is now part of most outsourcing contracts. In particular, interconnecting Service Providers' workflow systems such as Remedy, Service Manager, ServiceNow, with customers' workflow systems via Case/Service Exchanges (interfaces), see Figure 1, is a growing segment and often a central piece of larger contracts. These interfaces are highly visible, due to dependencies of other projects, thereby enabling Service Providers to service the overall contracts and as such it could affect customer satisfaction. The trustworthiness of services are as crucial factor determined by whether or not a potential consumers make use of a service offerings or even maintain them for long. These integrations start with requirement analysis phase that include fields mapping, data translation, protocol identification, detailing the product specification, defining validation and verification mechanisms, defining stories and test/use cases. This is followed by technical steps where protocols definition and fields/data mappings of Service Providers' work flow system such as Workflow System to those of the customer or their partners is conducted. After this phase, configuration and coding of interfaces on both side is done. Next test scripts definition and agreements are finalized, test oracles are written, and adequacy and coverage criteria are set. The next important step is a comprehensive multiple level of testing, utilizing agreed scripts, both by Service Providers and their customers/partners in their respective environments as well as those done together to test the integration of the systems. Functional and security aspects are tested thoroughly at this stage. The overall functional but specifically security testing is the focus of this paper. This testing cycle is finally complete conducting by User Acceptance Testing where the solution is scrutinized by the business teams to ensure the interfaces are behaving per process and business cases that were agreed during requirement phase, after this step the product is moved to production and is considered live. Loosely speaking, requirement analysis, configuration/coding and testing form the major steps in such interconnections. Therefore, it is important to conduct these steps in standard ways as much as possible. Innovative ways to reuse and reduce waste during the requirement analysis phase are required which are achieved by implementing requirements/definition documents that describe standards and incorporate use cases.

The testing techniques, the focus of this paper, needed for these projects require a significant amount of manual, non-execution based, efforts which typically result in inadequately tested scenarios unless creative techniques are

employed. The multiple iterative testing strategies, involving both non-execution based as well as execution based testing, are meant to reduce the required resources. We deploy both testing to specification (black-box testing) and testing to code (white box testing) [1].

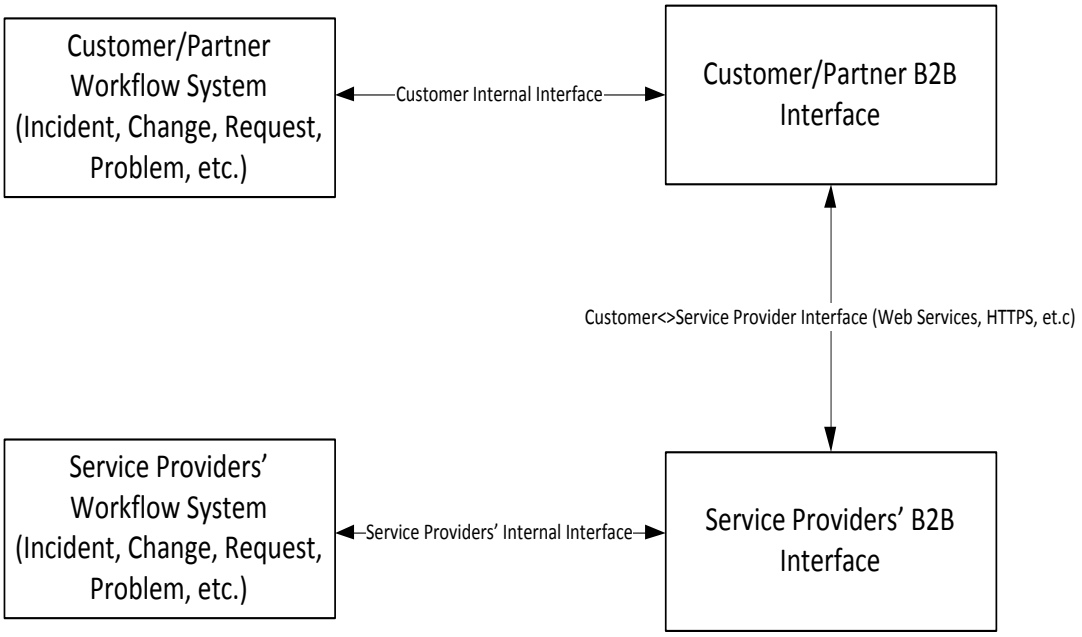


Figure 1: Workflow-to-Workflow Interconnectivity Capability for Different ITIL Modules

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tm="http://abc.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"xmlns:tns="http://xyz.Ws.v1/"
mlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" targetNamespace="http://xyz.Ws.v1/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
 <wsdl:types>
  <s:schema elementFormDefault="qualified" targetNamespace="http://xyz.Ws.v1/">
   <s:element name="Submit">
    <s:complexType>
     <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="Incident">
       <s:complexType>
        <s:sequence>
         <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="TransactionId" type="s:string" />
         <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="TransactionType" type="s:string" />
         <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="TrancSubType" type="s:string" />
         <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="SourceCaseId" type="s:string" />
         <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="DestinatiCaseId" type="s:string" />
         <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="CustomerId" type="s:string" />
         <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="PartnerId" type="s:string" />
         <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="ResultMessage" type="s:string" />
         <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="ServiceProvider" type="s:string" />
         <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="Description" type="s:string" />
         <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="CaseLog" type="s:string" />
         <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="LastEntry" type="s:string" />
         <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="Status" type="s:string" />
         <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="Impact" type="s:string" />
         <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="Urgency" type="s:string" />
         <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="Category" type="s:string" />
         <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="SubCategory" type="s:string" />
         <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="ProductType" type="s:string" />
         <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="ProblemType" type="s:string" />
         <s:element minOccurs="0" maxOccurs="unbounded" form="unqualified" name="ConfigurationItem">
.....
   </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Figure 2: WSDL

We deploy a closely coupled approach whereby requirements are tied to use cases which are tested thoroughly and issues are discovered early in the testing cycle. The discovery of bugs, product defects or the need for enhancements at later stages are exponentially more expensive and thereby a point of focus to be avoided or reduced significantly. The iterative nature of the development can make regression testing more efficient if conducted carefully. Another advantage of such test strategy is that developers (or testers) can use capture/replay tools such as WinRunner, to perform stress/performance testing when needed for very heavy and active interfaces [1-8].

II. FUNCTIONAL TESTING STRATEGY

As the most significant and the dominant success criterion for any software projects/products in the industry is the quality [9], this against the ever intense demand for fast turnover with much wanted cost efficiencies, requires enhanced development paradigms with testing taking center stage. It is not surprising then to note that the most time consuming and very crucial step in any software development, including integrations, is testing. In general, software testing is a process of providing inputs to software that is being tested and evaluating the produced results. The mechanism used to generate expected results is called an oracle. There are several approaches that we can adopt to generate, capture, and compare test results. Some of the common ways are:

A. Manual testing/verification of results by human- manual oracle (non-execution based);
B. Testing/Verifying specific values for known responses;
C. Testing/Verifying the consistency of generated values and end points;
D. Interface simulator to produce results;
E. Utilizing sampling of values to compare with independently generated expected results;
F. Automating the testing/verification for regression.

For functional testing autonomous system integrations, most of the above steps are utilized. These steps enhance the quality of the integrations and improve both schedule and cost. All of the tests utilized A through C, and by adding steps, D, E, F, DE, DF, EF, DEF we see improvements in both schedule and cost. Automating some aspects of this step is highly desirable [6-7, 10-11]. We showed in our earlier work the cost and schedule savings for functional testing for our proposed approach [6]. These projects need to be done in cost effective way with reasonable schedules. Testing is an essential step in these interconnections and efficient methods need to be deployed to achieve the desired results. We have shown a strategy that is deployed by a major Service Provider where significant savings (~17%-27.5% schedule and ~20%-25% cost reductions) were achieved by employing refined steps [6].

The computing speed has grown very rapidly and that coupled with low cost of memory, test cases can generate very large amounts of data. This makes the oracle data also massive which is needed for comparison. Integration work requires data comparison which must be incorporated into test cases. This requires that we add to test cases the error handling, capturing error results, as well as reporting differences. The nature of integration necessitates dependency on human oracles to verify test results, this can only be fruitful and efficient where the testers know the details of the code, and they are expected to know when the application misbehaves. Manual testing with tester as oracles has disadvantages that include increased costs and longer schedule durations and as such certain cycles of testing (in particular integration and regression testing) are automated as much as possible. New techniques and strategies are needed to reduce the test suites as well as detection of bugs/defects to be shifted to early test cycles (unit testing) in autonomous system integration projects. We have had some successes in this area by bringing the learning from previous work [6] and adjusting our overall testing strategy.

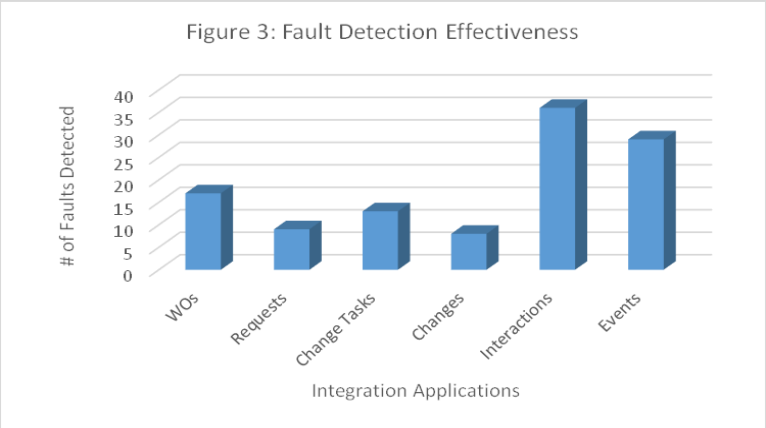
III. SECURITY TESTING STRATEGY

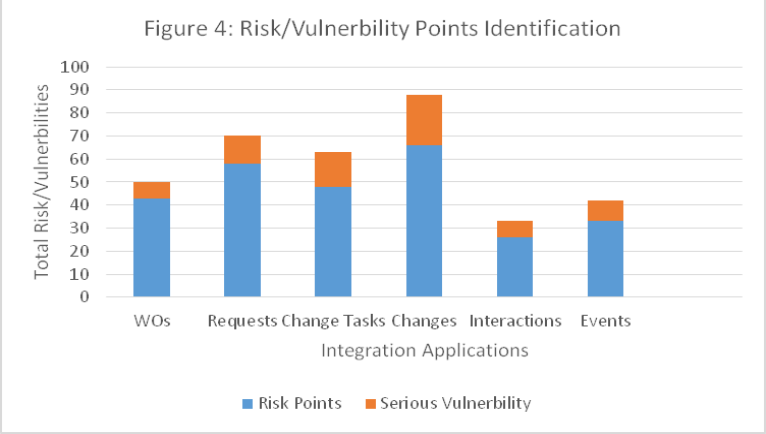
For autonomous system integrations, where workflow systems on the provider side is integrated with customer/partner side workflow system, the interconnections utilize XMLs via HTTPS POST, REST/RESTful, or SOAP mechanism. In these integrations, in addition to overall aspect, these mechanisms that are utilized need to be tested for proper functionality but also for security. The remaining of this paper will restrict, without loss of generality, the discussion to web services developed using SOAP protocol. The security aspect is tested using fuzzing technique using SoapUI and Burp Suite along with home grown tools. Web service is a software system designed to support interoperable system to system interactions over a network which make it ideal for workflow system interactions as those interactions come down to individual transactions that are sent in near-real time. We consider transactions for new exchange initiation for incident, request, request work order, change or change tasks. In addition, we consider acknowledgements, updates, close, cancel, misroute transactions for each of the above. Each transaction is based on a generic WSDL. The Web Services Description Language (WSDL) is an XML (Extensible Markup Language) based interface description language that is used for describing the functionality offered by a web service. A WSDL, see Figure 2, description of a web service provides a machine readable description of how the service can be called, what parameters it expects, and what data structures it returns. It thus serves a purpose that corresponds roughly to that of a method signature in a programming language. From the above sample you will notice that it mostly looks like any HTTP request except it has extra headers that are related to the SOAP protocol. The security testing thus takes the same approach that is after testing the functional aspects, using SoapUI, ensuring all the transactions are tested with all the fields and correct data translations, the security aspect is tested by creating a fuzz value file with values that might result in an invalid request or has a capability to trigger SQL injections, errors and overflows. We utilize Burp Suite which is an integrated platform for performing security testing. Burp Suite has multiple modules that are used to support the entire security testing process

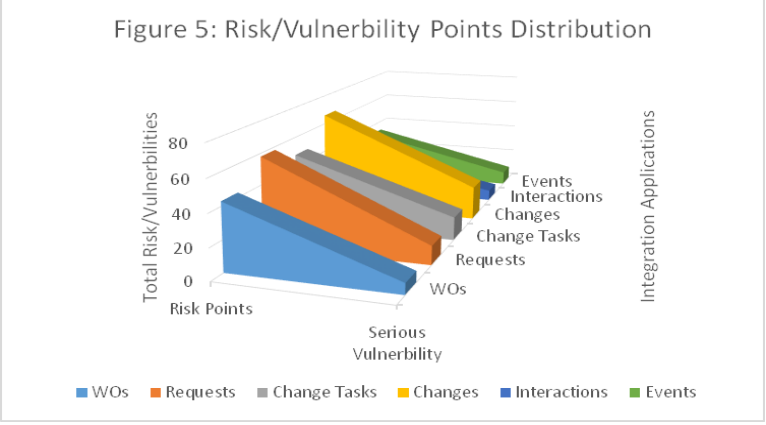
and we use Intruder and Repeater for this work. Burp Suite is flexible and give control over the testing where we can combine manual techniques with automation and is helpful from initial analysis of attach surface of our integration applications to highlighting the vulnerabilities. Using the original transaction and this fuzz file is fed to Burp Suite (by using Intruder) for fuzzing marked fields. The fields, which need to be tested, are manually selected. You cannot fuzz all the fields as that would exponentially increases the test cases and hence the cost and schedule a proper representative fields' selection is therefore chosen and run on a prioritized fashion so that we can ensure those with potential of uncovering high or more risk points are run earlier. Remember that our sample product types has one-to-several dozen fields of high importance that be fuzzed (which are manually selected.) The Intruder attack and responses are captured and analyzed to identify risk spots. We also use Burp Repeater to further test those identified responses with potential security risks. There are many other areas that come under the security testing umbrella such as interoperability testing whereby SOA allows integrations independent of coding environment (.NET, Java, PHP) or OS (Linux, Windows, etc.) To complete testing cycle, as part of interoperability testing, the areas that are needed to be focused are transport layer security, identity, and privacy/integrity interoperability. The other area is and performance testing. Both of these areas are not direct focus of this paper.

IV. RESULTS AND DISCUSSION

We have looked at two main results based on an array of experiments we run on six integration applications representing different levels of complexity from simple interaction, to simple work orders (WOs), even, task, request or changes in increasing order of complexity.







The application is used in a single or multiple transactions. The results that we studied are (1) the number of faults detected and (2) risk points identified, where some of the risk points represented vulnerabilities which could be exploited with high probability on an attack, by malware, or abused by other unintended entities. The column graph in Figure 3 shows the number of faults detected for all the test runs combined. In average we combined the results of 25 runs and normalized. The figure shows that our technique detected a large number of faults.

Figure 4 shows the number of risk points identified by our test suites. In both graphs, the x-axis shows the subject applications and the y-axis shows the number of faults detected and risk points by the test suite. In Figure 4, each column shows the total risk points and it shows on the vulnerabilities as well. Figure 5 gives another view highlighting the distribution from just risk identification to serious vulnerabilities for each application type. These results helped guide the testing to further focus test cases by fuzzing around the risk points but even more fuzzing fields that were related to identified vulnerabilities- as such acting as a prioritization mechanism. The fuzzing technique works for uncovering faults as well as risk points and serious vulnerabilities in simple integrations such as interactions and events, moderate work orders and tasks, and complex request and change workflow based integration applications.

V. CONCLUSION AND FUTURE WORK

We studied testing of a workflow based applications and complemented the testing with fuzzing mechanism to not only study the fault detection effectiveness but also guide testing to identify risk points and serious vulnerabilities to be further analysed. Interconnecting Service Providers' workflow systems such as Remedy, Service Manager, ServiceNow, with customers' workflow systems via integration interfaces is a growing segment and often a central piece of larger contracts. These interfaces are highly visible, due to dependencies of other projects to work smoothly, enabling Service Providers to service the contract and as such it could affect customer satisfaction positively (or unfortunately negatively if not done efficiently.)

These interfaces are also presenting security risk points and as such testing lifecycle needs to include security testing in addition to functional testing, which are often time intertwined in the development cycle, so that not only efficient and cost effective product with desired quality is delivered but also any vulnerabilities and security risks are studied in depth. In this paper we presented a new technique where we coupled the study of fault detection effectiveness with the detection of risk points and serious vulnerabilities using fuzzing as testing technique. The results are encouraging for a range of applications that we studied. The results were based on several academic lab runs and in corporate environment. We need to study these applications more and focus on each transaction for each application to further fine tune the risk identification and more in-depth analysis. Other class of SOA based application is another future studies. Extending this work to regression testing and in particular considering traditional versus agile development environments.

REFERENCES

[1] B. Beizer, **Software Testing Techniques**, (Van Nostrand Reinhold, New York, 2nd edition, 1990).

[2] M. Bozkurt, M. Harman, and Y. Hassoun., "**Testing web services: a survey**", Technical Report TR-10-01, Department of Computer Science, King's College London, April 2010

[3] W. T. Tsai, et al, "**Extending WSDL to facilitate Web services testing**", Proceedings of 7th IEEE HASE, pp. 171- 172, 2002.

[4] A.K. Onoma, W.T. Tsai, M. Poonawala, and H. Suganuma, "**Regression Testing in an Industrial Environment**", Communications of the ACM, Vol. 41, No. 5, May 1998, pp. 81-86.

[5] R. Paul, "**End-to-End Integration Testing: Evaluating Software Quality in a Complex System**", Proc. of Assurance System Conference, Tokyo, Japan, 2001, pp. 1-12.

[6] S. Faizullah, "**Some Practical Considerations and a Methodology For Testing Autonomous System Integrations**", International Journal of Software Engineering & Applications (IJSEA), Vol.5, No. 1, Jan. 2014, pp 1-8.

[7] S. Faizullah, "**Regression Testing as a Quality Software Development Tool- Our Experience in Large Projects in the Industry**", Key Note Talk at Fourth International Workshop on Regression Testing (Regression 2014), IEEE International Conference on Software Testing, Verification and Validation (ICST) 2014, Cleveland, Ohio, USA, March 31st - April 4th, 2014

[8] S. Faizullah, "**Testing Autonomous System Integrations**", Invited Talk at Umm al Quraa University (UQU), Makkah Mukkaramah, KSA, Mar. 2014.

[9] M. Younus, "**Fundamentals of SOA Security Testing**," Service Technology Magazine Issue LIX, Jan. 2014, pp 1-6.

[10] P. Godefroid, M.Y. Levin, D. Molnar, "**Automated Whitebox Fuzz Testing**," In Proceedings of Network and Distributed Systems Security, Feb. 2008, pp. 151-166.

[11] S. Faizullah, "**Rapidly Evolving Research Area- Testing Software Testing**", Invited Talk at COMSATS Institute of Information Technology (CIIT), Abbottabad, Pakistan, 2009.

[12] S. Faizullah, "**Trends in Testing Software Testing- A Case in Focus**", Talk at Frontiers in Technology (FIT) 2010, Islamabad, Pakistan, Dec 21-23, 2010.

ABOUT AUTHOR

Safi Faizullah received his Ph.D. in Computer Science from Rutgers University, New Brunswick, New Jersey, USA in 2002. He also received MS and M. Phil. Degrees in Computer Science from Rutgers University, New Brunswick, New Jersey, USA in 2000 and 2001, respectively. Dr. Faizullah also earned his BS and MS degrees in Information and Computer Science from KFUPM, Dhahran, KSA in 1991 and 1994, respectively. His research interests are in computer networks, mobile computing, wireless networks, distributed and enterprise systems. He has authored over twenty refereed journals and conference papers. Dr. Faizullah works for Hewlett-Packard and he is a Visiting Research Professor of Computer Science at Rutgers University. He is a member of IEEE, SCIEI, PMI and ACM.

