Unit and Integration Testing of Modular Cloud Services

Conference Paper · March 2017

DOI: 10.1109/AINA.2017.57

CITATION
1

READS
10

4 authors, including:

Stelios Sotiriadis
University of Toronto
76 PUBLICATIONS **529** CITATIONS
SEE PROFILE

Euripides Petrakis
Technical University of Crete
122 PUBLICATIONS **2,344** CITATIONS
SEE PROFILE

Nik Bessis
Edge Hill University
199 PUBLICATIONS **1,246** CITATIONS
SEE PROFILE

Unit and Integration Testing of Modular Cloud Services

Stelios Sotiriadis[1,3], Andrus Lehmets[2], Euripides G.M. Petrakis[3], Nik Bessis[4]
[1]Computer Engineering Research Group, University of Toronto, Toronto, Canada
[2]Andrus Lehtmets, Elvior, Estonia
[3]School of Electronic and Computer Engineering, Technical University of Crete, Chania Greece
[4]Edge Hill Univerity, Ormskirk, UK
s.sotiriadis@utoronto.ca, andrus.lehtmets@elvior.ee, petrakis@intelligence.tuc.gr, Nik.Bessis@edgehill.ac.uk

***Abstract*— Cloud computing and the future Internet concept highlight new requirements for the software engineering phases including testing and validation of modular web services. A major reason is because cloud applications are developed by services belonging to different providers, thus making software testing a really challenging issue. In this work, we propose a testing methodology that includes two fold testing actions; a unit testing of cloud service APIs following white and black box techniques and an integration testing strategy by identifying services that could interface with each other. In addition, we present the Elvior TestCast T3 (TTCN-3) testing tool for automation of use case testing. We demonstrate the results of the methodology when applied to different cloud services and we present a discussion of our conclusions for a real world use case, in which we applied this methodology.**

Keywords—Cloud computing; Cloud services; Cloud service testing; TestCast; TTCN-3

I. INTRODUCTION

The growth of Internet characterizes the so-called Future Internet (FI) concept [34] that allows development of innovative applications from modular services belonging to different owners and developers known as cloud enablers [20, 23]. Modularity in cloud computing includes already deployed services that provide open specifications and are available for utilization using APIs (i.e. RESTFul [9]). While these solutions highlight innovation and promotion of a new easy-going development method, software engineering processes are becoming more and more complex. This is because such enablers have distinct features that impact several different research fields, including software testing [29]. These include execution over virtualized resources that can be highly scalable and elastic, for instance, the users can increase their computational capacities and share common physical resources with other cloud applications or services, thus supporting multi-tenancy. Also such services provide interfaces using APIs that allow combination and remote management as well as a service could be developed based on another.

So, cloud services testing introduces a new set of challenges [27] and requirements that have a direct impact on system engineering processes [19]. The testing methodology does not only involve the validation of a software internal processes, but also its APIs and the interactions with other services. The complexity factor is also increased due to the fact that software engineers base their design on abstract and high level use case models. Thus it becomes a hurdle to design test cases that are based (a) on the application use case requirements (part of the requirements engineering process [22]) and (b) on the services use case requirements' (that are functionalities of the cloud enablers). Based on this discussion, in this work we aim to perform testing (unit and integration) of cloud enablers that are composed by other cloud services. The challenge is to conceptualize testing processes across a unique methodology. We also characterize key features of cloud services testing methodology according to various aspects e.g. elasticity, security, performance, compatibility, API integration and multi-tenancy.

We apply our methodology in real world cloud services deployed in OpenStack system [36] and we present an empirical study for white and black box testing of FIWARE[1] Generic Enablers (GEs) and FI-STAR[2] Specific Enablers (SEs) as in [33]. FI-WARE offers specifications and tools to build an FI application from a variety of GEs with fundamental functionalities, interfaces and APIs. In parallel, FI-STAR establishes early trials in the Health Care domain building on Future Internet (FI) technology. To automate the testing process we propose the utilization of the Elvior TestCast toolbased on the Testing And Test Control Notation (TTCN-3) standard [24, 25]. The tool offers key features to the overall methodology including a test editor, compiler, debugger, test execution suite and viewer. By using it, cloud software testers can easily create unit tests according to standards adoption. Based on this discussion, next Section II presents the motivation of our work, Section III the review of the related approaches, Section IV the proposed testing methodology of cloud services and Section 5V the conclusions and future research directions.

II. MOTIVATION

There are many cloud providers offering modular services i.e. IBM microservices [3], FIWARE etc. that allow large complex applications to be easily created and deployed

[1] https://www.fiware.org
[2] http://www.fi-star.eu
[3] http://www.redbooks.ibm.com/redbooks/pdfs/sg248275.pdf

independently. Today, there are new requirements related to the software engineering processes of cloud enablers since there are significant changes taking place as to how to test the new software running on the latest cloud platforms [31]. Especially in case of testing this refers to the gap between software developers and software test engineers. There are three key differentiations among traditional and cloud enablers testing and these are related with (a) scalability of virtual resources, (b) multi-tenancy efficiency that refers concurrent users and (c) dynamic reconfiguration of services [35]. As a result, the cloud testing models have to support different kinds of requirements [26], thus, tend to become more and more complex. In more detail, traditional applications are firstly designed and then tested, however, today SaaS involves on-the-self ready-made software. Also, it involves frequently updates that are usually with zero downtime. A cloud service functional testing includes a series of characteristics such as unit testing in an automated way thus to ensure high volume of test cases coverage. Also, it requires testing plan preparation according to the cloud enablers and their inter-dependencies.

This work is motivated by work of [30], where authors suggest that there is a lack of research papers addressing new issues, challenges, and needs in SaaS testing. We focus on cloud services testing and we propose a methodology enabling efficient unit and integration testing of modular services. We define a testing design approach that can be summarized as (a) High level testing design strategy: This includes the top-down approach (that is according to service specification) and the bottom-up approach (that is according to functional building blocks of applications) and (b) Software testing strategy: This includes the white box (internal structures are known to the tester) and the black box testing. Here each strategy includes unit and integration test cases that could be applied to the high level testing design strategy.

III. Review of Related Approaches

This section presents an analysis of cloud testing approaches and solutions. In [16] authors present a graph theoretical model aiming to describe the concepts of a cloud as a graph. They utilize predicate-enabled subclouds in order to show interactions among clouds and to be captured and represented by a model that supports formal analysis. This is a theoretical study focusing on cloud computation and it does not involve concurrency of services or modularity in terms of service encapsulation into another. In [17] discuss the concept of software Testing as a Service (TaaS) [28]. The authors discuss that this clearly puts testing out of the sequential line of the software development process as discussed by [1]. Authors also suggest that TaaS includes two aspects, a service dedicated to developers and one to the end-users from the perspective of quality assurance.

In [2] a survey is presented to characterize software testing in cloud. They include the Testing as a Service (TaaS) and conclude to key risks that associated with cloud testing. These are security, lack of standards, infrastructure and planning. At last authors conclude to a key research issues that are (a) how to achieve validation of the quality of cloud application and services at all levels and (b) how to manage testing for cloud application processes belonging to different clouds. Both issues highlight the research questions of this study. In [3] a discussion is presented to compare conventional and cloud testing approaches. The works concludes to similar results regarding efficient planning through effective methodologies, adoption of standards and execution of tests in real-time test environment. In [4] authors discuss the SOA at various testing levels including integration testing. In particular, they present the challenges of the service dynamic bindings stressing out the problem of polymorphism in SOA and the complexity of testing a service against all the possible endpoints identifying a major research question of our study, that testing could happen too late in the overall application or service development life-cycle, thus there is clear gap regarding to a successful testing design.

In [1] authors present a tutorial on cloud testing and cloud-based application testing. They also present the need for integration testing in clouds by denoting that “not much research results have been applied in the real engineering practice”. In this work we take advantage of the existing experiences in testing a large scale FP7 project in order to characterize a real case testing methodology. In [5] authors present ideas for testing Web Services by using a design by contract that adds behavioral information to its specification. They introduce the notions of provided contracts (describe the behavior offered) and required contracts (describe the behavior needed) by a Web Service. This highlights an important issue of this work, the classification between test use case development according to top-down (encapsulating the behavior offered) and bottom-up (encapsulating the behavior needed) approaches. In [6] authors present an analysis of the software testing as an online service based on the question “What are the conditions that influence testing of online services?” by highlighting the research questions of this study. The authors classify these conditions according to TaaS.

In [7] authors discuss the migration of testing into the cloud. They suggest that a suitable testing will need to involve unit and regression testing, high volume of test cases and availability of interfaces for test automation. In [8], authors present a decision framework solution called SMART-T that supports migration of software testing actions into the cloud. The authors include a distributed environment called “HadoopUnit” that allows the cloud based concurrent execution of test cases and a sequence of use case scenarios to demonstrate the use of the framework and its environment. In [9] authors present a practical example of an application namely as QuickCheck that focuses on the testing of REST Web services. They further present a technique that allows adaptation of this model to perform testing for specific scenarios over HTTP. In [10] authors present an automated test harness for a cloud service, with the delivery of TaaS. They conclude that from a technical perspective there is a clear need for a new wave of development and testing tools that will address the requirements of new cloud computing platforms. In [11], the author presents a review of approaches of cloud testing and suggests that this area is still maturing. Finally, he claims that cloud testing is an important issue of the cloud software development life cycle.

Next, we present a discussion of the most well know tools and solutions. In [12], authors present Cloud9, a software

testing service that aims to reduce the resource-intensive and labor intensive nature of high-quality software testing. The work focuses on demand software testing that support the symbolic execution. In [13] authors propose D-Cloud, a software-testing environment that uses cloud computing technology that focuses only on virtual machines with fault injection facility [1]. In [18] authors present the Apex framework that enables the testing utilizing build-in support for test creation and execution in the Force.com platform. The authors claim automation of unit testing based on simulations, however do not include integration testing characteristics. In [14] authors present the CloudAnalyst tool (based on CloudSim) that simulates large-scale cloud applications and includes insides about testing various deployment configurations. In [21] the SimIC is presented that allows simulations of clouds and inter-clouds. In [Gao et al. 2012] proposes a TaaS infrastructure presenting a cloud-based TaaS environment with tools (the so-called CTaaS) that aims to meet the needs of SaaS testing, performance and scalability evaluation. Their study presents a simulation analysis of results. Finally, in [15], authors present a York Extensible Testing Infrastructure (YETI) that is a cloud testing solution. [11] suggest that YETI could lead to performance and security problems if it is executed into a single computer. Next we present the web based testing tools.

Selenium Testing[4] offers a test suite that allows testing on various popular languages (PHP, Python etc.) for local and firewalled applications. The *SOASTA*[5] provides products for real-user monitoring, mobile testing and load and performance testing. The *iGATE Patni*[6] is a TaaS cloud based framework for test automation supporting dynamic scalability at low-cost. The *Janova*[7] provides a suite for automated testing in the cloud. The tool allows transformation of simple rules that are derived by the requirements into tests. The *iTKO*[8] allows functional testing of dynamic web sites and rich applications. It includes depth in test coverage and high level of reusability and extensibility. The *LoadImpact*[9] is a testing suite that allows full control on API and SDKs.

A particular interesting standard is the Testing And Test Control Notation Version 3 (TTCN-3) [10] that offers global standardization, independency with regards to the execution environment and flexibility in designing and maintaining test software. Classical black box testing method is used for test automation of implementations using RESTful APIs and TTCN-3 as standardized test automation technology is used for building test environments. There are following cornerstones for using TTCN-3 [24] for testing RESTful APIs as follows. TTCN-3 is a mature technology and has been widely used for black-box testing and it fits very well (but not limited) for client-server type implementations (which is one of the REST constraints). There are several TTCN-3 compiler providers in the market including the 2 leading commercial TTCN-3 tool providers that are Elvior TestCast T3 [11] and Testingtech TTWorkbench [12], both being active in ETSI at TTCN-3 standardization. Ericsson uses TTCN-3 widely in its internal projects and their tool Titan became open source in the beginning of 2015. Other TTCN-3 tools are OpenTTCN Tester[13], IBM Rational System Tester[14] and Devoteam TTCN-3 toolbox[15].

To the best of our knowledge, the proposed Elvior TestCast T3 is the only tool on the market which compiles TTCN-3 scripts directly to binaries while all other tools have intermediate step i.e. to compile TTCN-3 into other high level programming language (C, C++ or Java) and apply appropriate compiler to build test executable. Therefore, TestCast T3 gives shorter learning curve and more natural way to achieve test cases execution during test development. Other unique features are (a) it supports C# mapping (additional for C, C++ and Java) for TRI and TCI interfaces and (b) so SA (system adapter) and codecs developers can use all benefits of .NET framework. TestCast T3 and its TRI framework is built so that SA is always separate program from test executable (while other tools often link SA with test executable). TestCast T3 also communicates with SA always over TCP/IP – this makes test environment deployment very flexible.

Further to these, a new requirement that in particular arises in the concept of the Next Generation Service Interfaces (NGSI) information model. Elvior TestCast T3 allows test cases of NGSI models. This is context management information model, developed by Open Mobile Alliance (OMA) [16] that includes NGSI abstract interfaces of context based APIs. OMA is the wireless industry's starting point for the development of mobile service enabler specifications, which support the creation of interoperable end-to-end mobile services. The Context Management APIs provides interfaces in order to manage context information about entities and access to the available context information about the entities. It supports actions to register and retrieve information and update and query context. It provides two interfaces the NGSI-9 that is responsible for register, discover subscribe and notify and NGSI-10 that is responsible for context information update, query, subscription and notification. NGSI is used widely to include easy representation of the Internet of Things (IoT) devices.

Different from the aforementioned works, we focus on cloud enablers that are decentralized, modular and are the building blocks of other applications or services. To sum up the features of this article are summarized as follows. The testing methodology of cloud services includes bottom-up and top-down approach (that involves requirements coverage based on testing) as well as unit and integration (test plan and phases) strategy for test case design. Different from existing solutions we focus on the issue of developing cloud applications based on services that again might be result of a second level

[4] Selenium Testing: https://saucelabs.com
[5] SOASTA: http://www.soasta.com
[6] iGATE Patni: http://igatepatni.com
[7] Janova: http://www.janova.us
[8] iTKO: http://www.itko.com/solutions/functional.jsp
[9] LoadImpact: https://loadimpact.com
[10] TTCN-3: http://www.ttcn-3.org

[11] TestCast T3: http://www.elvior.com
[12] Testingtech TTWorkbench: www.testingtech.com
[13] OpenTTCN: http://www.openttcn.com
[14] IBM: http://www.ibm.com
[15] Devoteam TTCN-3 toolbox: http://www.devoteam.de
[16] OMA: http://technical.openmobilealliance.org

development (i.e. composed by other cloud services). Also, the testing execution and software validation using the TTCN-3 standards ensures diversity of testing exposure metrics, adoptions of standards and test cases independency. Different from existing tools, TestCast T3 tool that is utilized in this study includes ETSI, OMA, NGSI-9/10 and FIWARE compliance.

IV. Testing Methodology of Cloud Services

The methodology includes the testing requirements analysis and the test cases analysis of cloud enablers. Figure 1 demonstrates the IEEE Standard for software and system test documentation (IEEE829 [17]) standard processes and the identified new requirements. In particular, it determines test processes for the development of products that will need to conform to the requirements according to the user needs for different integrity levels (test plan, test design, test execution and summary).

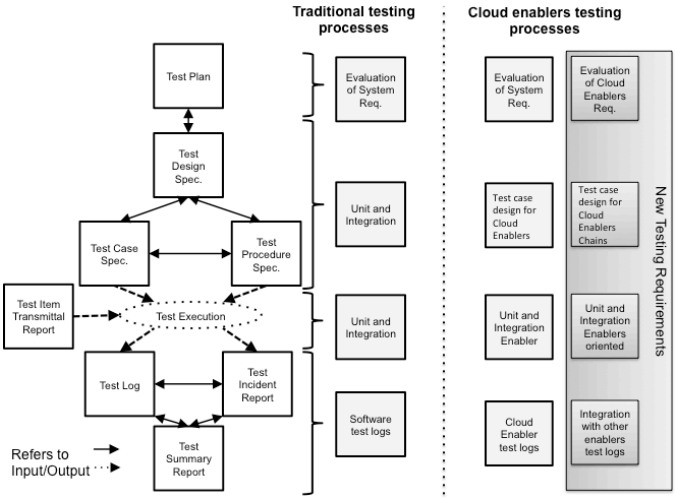

Fig. 1: The comparison of traditional and cloud enablers testing processes (divided into test plan, design execution, and reporting) according to IEEE829 standard

The following sections include the enablers unit and integration testing strategy (Section A) and the integration requirements analysis strategy (bottom up and top down in Section B). The testing preparation strategy defines the testing schedule and plan for black box testing of cloud enablers including conceptualization for unit and integration testing. Firstly, the unit testing allows cloud services' testers to execute various tests with different input parameters to the operations and interfaces defined in service specification documents and manuals. The modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are proper developed. The integration testing will allow combination and test execution of linked cloud enablers in order to test key functionalities.

The testing starts with the unit testing preparation activity where each cloud enabler tester defines the unit test and unit test cases so the unit test defect management process will allow bug tracking and fixing. Then the integration testing includes preparation, execution and defection management process regarding the inter-dependencies of enablers. At this stage, the aim is to exploit the Functional Building Blocks (FBBs) of the application use case to map the key functionalities to cloud enabler dependency chains. In more detail, the unit testing includes the testing activity of the cloud enablers by focusing on the evaluating of their interfaces. Developers perform unit testing during and after development. In particular, each enabler tester will need to provide an overview section of its description based on the unit testing matrix that includes the unit testing enabler Status, the overview of the testing actions, the unit testing activities, the unit test cases description, and the description of unit test cases execution.

The integration testing, describes the integration testing analysis for enablers based on their features (top-down) and the detailed description of the identified chains by use case scenario (bottom-up); both forming the overall integration testing chains for back end SEs. This part is the level of the software testing process where individual units (various cloud enablers) are combined and tested as a group that encapsulates a particular operation named as integration chain. The Integration Testing Matrix (ITM) includes the steps required. This section also demonstrates a detailed description of integration chain identification.

A. *Unit Testing*

This section demonstrates the unit testing strategy and the various matrices for testing analysis. The Unit Testing Matrix (UTM) structure is shown in Table 1 that has been developed to provide a template for characterization of the testing actions. Table 2 demonstrates a template of the UTM that includes the unit test number, the interface name the input and output data and the unit test case number.

Table 1. The Unit Testing Matrix (UTM)

Definition of the Unit Tests	Describes the features to test, including a description of how they will be tested.
Definition of the Interfaces to be tested	Describe the definition of exposed interfaces and their data types and parameters.
Unit Test Cases descriptions	These are in the form of tables that demonstrate scenario cases that include interface class name, interface name, operation/function name, input and system output and evaluation of the expected and actual result.

Table 2. Template of the Unit Test Matrix

Unit Test	Interface	Input	Output	Unit Test Cases
UT#	/interface_ name	Call input: data http://URL/interf ace_name/data	Data format	UTC#

Table 3 demonstrates the Unit Test Case Matrix (UTCM) template including all the required information regarding the interfaces, results, invalid data input and information about the instance data (e.g. provider).

[17] https://standards.ieee.org/findstds/standard/829-2008.html

Table 3. Template of the Unit Test Case Matrix

UTC#	
Description	Description of the UTC for appropriate UT, this involves further documentation of the input/information model set.
Unit Test:	UT number.
Class Name:	The class name of the interface.
Interface name:	The interface name.
Operation:	Description of the operation/functionality of the interface.
Input:	Input data and information model.
Output:	Resulting output.
Expected Result:	Description of the testing results (result as expected or failure)
Actual Result:	Required results that in the case of failure are the expected results.
Invalid data input:	Description of invalid input set (intentionally submission for testing reasons)
Details:	Description of the testing environment and/or software (e.g. browser).
Other remarks:	Description of notes.
Screenshot	Illustration of results.
Deployed Instance:	Endpoint of the cloud enabler.
	Information about the cloud provider.

To achieve white and block box testing the strategy includes two approaches of the unit testing for ensuring test coverage; the unit test by enabler owner (white box) and the unit test by an external tester that uses the Elvior TestCast T3 tool (black box). These are described as follows.

(a) Unit testing of enablers executed by the developer: In this case owners define the unit tests and unit test cases based on the design and specification documents. The aim is to test the interfaces against a set of various inputs so to evaluate functionality. Each SE owner provides details using the UTM and the UTCM matrices.

(b) Unit Testing of enablers using the Elvior TestCast T3: We expected that this would endorse the actions of testing performed by the enabler owners and highlight new requirements e.g. adoption of standards and automatize testing activities. In particular, an external tester provides several of UTs and UTCs for enablers based on the description of their intergaces. The tool offers particular advantages such as testing based on standards, modular testing language specifically designed for testing and supported by an active community, support of automated and distributed testing and reusability of test functionalities in different contexts.

The rationality behind the relationship of enabler owner and external tester is based on the following facts.

(a) Unit testing has been enriched so to achieve better UT and UTC coverage.

(b) Enablers' verification is according to functionality and standards that has been and double-checked according to the test executions.

(c) An automatized process is achieved that allows easier and more efficient testing (e.g. increase test cases of input data).

(d) Increased number of test cases according to different scenarios (e.g. regarding sequential testing including different executions of test based on various orders and flows).

B. Integration Testing

This section presents the analysis of the high level methodology that is comprised by the top down and bottom up method for integration testing. The aim is to define a functional hierarchy of the testing activity by achieving efficient planning, analysis and design of test cases with full coverage. The method is based on the functional decomposition of either (a) use cases requirements or (b) the cloud enablers' specifications, by broken each of which into several operations. Next we describe the two methods. The Top Down includes with the analysis of their cloud enablers and their features and interfaces that generate characteristics on which the interaction chains are based upon. This integration testing method focuses on the design of testing cases according to the top layer of the information that is related with the cloud enablers' specification and configuration documents as derived from the cloud software engineers. It starts with the basic operations and it gradually maps functionality and interfacing of cloud enablers according to the enabler specifications. Figure 2 demonstrates the Top Down steps for integration tests execution.

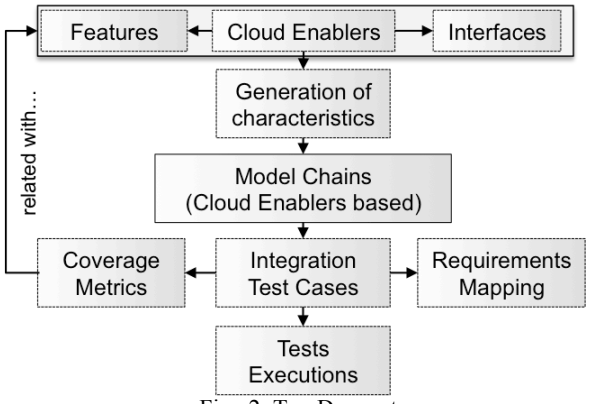

Fig. 2: Top Down steps

The Bottom Up includes with the analysis of high-level use cases architecture and its features to the mapping of FBBs to enablers and their relationships. This integration testing method focuses on the identification of use case requirements according to their abstract descriptions. The method starts with the use case analysis and characterization of the FBBs that model the enabler chains. Then the process involves the gradually mapping of FBBs to cloud enablers' functionalities and eventually to chains of inter-related enablers that encapsulate the full FBB operation. Figure 3 shows the Bottom Up steps.

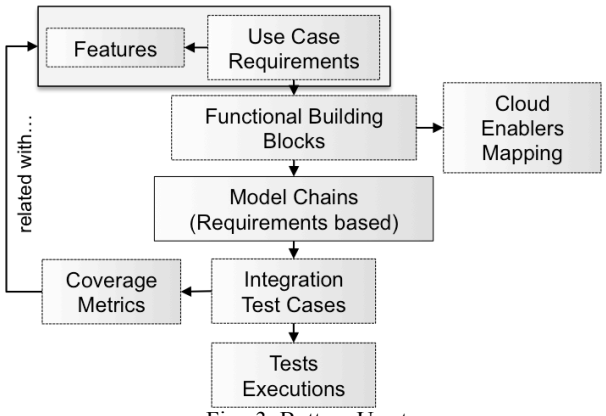

Fig. 3: Bottom Up steps

In Top-Down integration method, test cases are defined in terms of enablers' functionality (so deriving from functional requirements). In many cases this could be a hurdle since testers will require having extensive knowledge of enablers operations, while mapping of operations to FBBs could be proved to be highly technical for testers. A solution could be the application of a modified top-down testing strategy in which testing involves each layer of the system decomposition individually before merging the layers. In contrast, the Bottom-Up integration method involves mapping of FBBs to already offered operations, an action that requires refinement among the most and less important features. In this work, we propose that both methods should be used in order to ensure integration testing coverage. The next steps demonstrate the top down solution.

- According to the integration strategy, select the cloud services to be tested that include the basic operations of the use case.
- Set cloud services together based on their interfaces (according specifications) and do any preliminary fix-up that will make the integration test operational.
- The functional testing analysis will include definition of FBBs of test cases that will cover interactions of cloud enablers according to their architecture. This will include a structural composition testing of chains.
- Execute a series of integration tests based on various input/outputs and keep their records.

Table 4 demonstrates the Top Down Integration Testing Matrix (TD-ITM) that encompasses integration testing configuration such as interfaces, SEs, GEs and results.

Table 4. Top Down Approach Integration Testing Matrix

Integration Testing Definition/execution	Description of Testing Use Case
Integration Testing IT#	Interactive chain.
Description	Description of the integration chain.
Class Name:	The name of the class.
Interface name:	The name of the interface.
Operation	The description of the operation.
Interacted SE:	The name of the interacted SE.
Interacted GE:	The name of the interacted GE.
Input:	The input data and/or information model description.
Output:	The output data and/or information model description.
Expected Result:	The output of the test
Actual Result:	The actual results.
Invalid data input:	Description of the invalid input
Other Remarks	Remarks.
Deployed Instances IPs	Description of deployed instances.
Time/Date	Date and time information.

Next we demonstrate the steps followed in the bottom up solution in order to produce integration tests.

- Users should select the use cases to be tested that include the use case basic functionalities.
- Produce a map of FBBs to SaaS operations and identify candidate enablers that encapsulate required functionality.
- The functional testing analysis includes definition of test cases that will cover use case requirements to the associated cloud enablers. This includes a structural composition testing of chains.
- Execute a series of integration tests based on various input/outputs and keep their records.

Bottom up integration test cases are constructed to test functionality and linkage of cloud enablers according to the application use case FBBs. At this stage, the application tester identifies all the enabler couples that will need to put under test and follows the next steps. Table 5 demonstrate the dependency chains template.

Table 5: FBBs and their associated chains of enablers' dependencies

Features	Interactions
FBB_1	$Enabler_1 \rightarrow Enabler_2 \rightarrow \ldots$
…	…

Next (Table 6), the simplification process includes deleting of dependencies that are repeated to other chains. For example, in the case of Table 5, the pair Enabler2 → Enabler3 will be removed.

Table 6: Simplification process of FBBs chains

Features	Interactions
FBB_1	$Enabler_1 \rightarrow Enabler_2 \rightarrow \ldots$
…	…

Finally, each chain is described by a dependency (pairs of cloud enablers) and according to a specific testing order.

C. General overview of the tool and TTCN-3 ecosystem

The Elvior TestCast is a full featured TTCN-3 tool and can be effectively use for automated testing of cloud based systems. TestCast is ideal for incremental development where users can test specific systems and features separately as well

as the entire system as a whole. Figure 4 demonstrates the Principal architecture of TTCN-3 test environment. In detail, the System Adapter (SA) is used for communication of test tool (test executable) with System Under Test (SUT). The interface between test tool and SA is standardized and called TRI (TTCN-3 Runtime Interface) – SA is usually a piece of software that can be written in different languages such as C, C++, C#, Java, mainly depending on test tool. The other important interface is TCI (TTCN-3 Control Interface) that is used mainly for implementing external codecs again either in C, C++, C# or Java. Codecs (and their tool support) is essential part of TTCN-3 test environments because its type system is not bind to any binary representation. It is entirely up to the test tool and its codecs to ensure encoding and decoding of data in appropriate format.

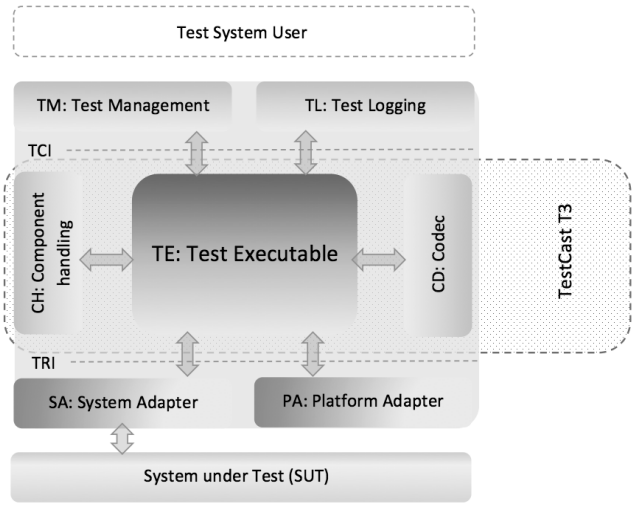

Fig. 4 Principal architecture of TTCN-3 test environment.

The characteristics of the Elvior TTCN-3 tool are as follows. The tool is developed in C# and could be run in Microsoft Windows and Linux platforms. Also, it has a user-friendly graphical IDE for test development and management. It has a native compiler that supports all TTCN-3 standards (up to TTCN-3:2015) and includes a native TTCN-3 debugger. The tool supports TTCN-3 test execution and XSD import. It has built in codecs such as textual, binary, XML (via XSD schemas), TCI XML, ASN.1 and supports TRI and TCI mapping for C, C++, C# and Java. Finally, it offers a rich TTCN-3 editor and test suite viewer and a enriched logging (textual and graphical views) and logs analyzing capabilities. The tool provides a test environment for RESTful interfaces testing. The roles of actors in the TestCast Tool BBT testing environment are (a) System Under Test (SUT): The implementation using the RESTful API, (b)TestCast T3 test tool for TTCN-3 test development and execution (it is responsible for execution of TTCN-3 test scripts), (c) HTTP SA (developed in C#): It completes communication between SUT and test tool. It acts as a server or client depending on test case needs and is controlled by TTCN-3 test script, (d) XML TestCast T3 built in codec and the external JSON codec (developed in C#): Codecs are responsible for encoding and decoding messages sent/received to/from SUT.

For RESTful interface a generic TTCN-3 framework for sending and receiving HTTP messages was created. The scripts are written and executed in TestCast T3, which sends messages to the HTTP adapter. The System Adapter creates HTTP messages based on the received information from TestCast and sends each of which to the SUT. TestCast built in XML and external JSON codecs; any received message from the SUT is decoded and sent to the TestCast via the System Adapter. In practice, the following actions are executed: i) message (hex sequence) is received from the SUT via SA to TestCast, ii) TestCast (TTCN-3 test executable) reads this hex sequence from TTCN-3 port and iii) the message is decoded by TestCast and can be used in TTCN-3 script. TTCN-3 libraries include common types and templates for forming messages for RESTful APIs.

V. CONCLUSIONS

This study aimed to present the empirical analysis of our experiences in testing cloud enablers. The contribution of this work to existing literature involves the testing methodology and test design matrices for cloud services. Different from existing solutions we focus on the test case development of cloud applications based on services that again might be result of a second level development (e.g. composed by 3rd party services). In addition, we present the Elvior TestCast T3 tool that includes ETSI, OMA, NGSI-9/10 and FIWARE compliance. The proposed solution assists cloud enabler testers to rapidly isolate errors and failures based on the virtualization of services so to identify dependencies with other components. Different from other solutions, the approach provides a complete end-to-end transaction that integrates chains of Applications, SEs and GEs. To demonstrate effectiveness, we presented the real world case scenario of FIWARE and FI-STAR projects. We emphasized the adaption of the proposed methodology and tools to the production of test cases; analysis of data and automation of processes based on real world use case requirements [33] for cloud-based healthcare applications. The future research steps include the testing of more complex scenarios regarding performance metrics and multi-tenancy. In addition, we aim to increase the number of TestCast test cases.

REFERENCES

[1] Jerry Gao, K. Manjula, P. Roopa, and E. Sumalatha, Xiaoying Bai, W. T. Tsai, Tadahiro Uehara. 2012. A Cloud-Based TaaS Infrastructure with Tools for SaaS Validation, Performance and Scalability Evaluation, 2012 IEEE 4th International Conference on Cloud Computing Technology and Science, pp.464,471

[2] A.Vanitha Katherine and K. Alagarsamy. 2012a. Software Testing in Cloud Platform: A Survey, *International Journal of Computer Applications (0975 – 8887)*, Volume 46– No.6, pp.21,25, May 2012 21

[3] A.Vanitha Katherine, Dr. K. Alagarsamy, 2012b. Conventional Software Testing Vs. Cloud Testing, *International Journal Of Scientific & Engineering Research*, Volume 3, Issue 9, pp.1,5, Spetember-2012 1 ISSN 2229-5518

[4] Gerardo Canfora and Massimiliano Di Penta. 2009 Service-oriented architectures testing: A survey, Software Engineering, pp.78,105

[5] Reiko Heckel and Marc Lohmann. 2005. Towards Contract-based Testing of Web Services, *Electronic Notes in Theoretical Computer Science* 116 (2005) 145–156

[6] Leah Muthoni Riungu, Ossi Taipale, Kari Smolander. 2010. "Software Testing as an Online Service: Observations from Practice," In Third International Conference on Software Testing, Verification, and Validation Workshops (ICSTW), pp.418,423

[7] Tauhida Parveen and Scott Tilley. 2010. "When to Migrate Software Testing to the Cloud?," *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on* , vol., no., pp.424,427, 6-10 April 2010 doi: 10.1109/ICSTW.2010.77

[8] Tilley, Scott and Parveen, Tauhida. 2012. Software Testing in the Cloud, *Software Testing in the Cloud Migration and Execution*, SpringerBriefs in Computer Science

[9] Pablo Lamela Seijas, Huiqing Li, Simon Thompson. 2013. Towards Property-Based Testing of RESTful Web Services. *Proceedings of the twelfth ACM SIGPLAN workshop on Erlang*, pp. 77,78

[10] Tariq M. King and Annaji Sharma Ganti. 2010. Migrating Autonomic Self-Testing to the Cloud, Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on, pp. 438,443, 6-10 April 2010

[11] Sergiy Vilkomir. 2012. Cloud Testing: A state-of-the-art review, Information and Security. An international journal, Vol28, No.2, pp.213,222

[12] Liviu Ciortea, Cristian Zamfir, Stefan Bucur, Vitaly Chipounov, George Candea. 2010 Cloud9: A Software Testing Service, ACM SIGOPS Operating Systems Review archive, Volume 43 Issue 4, January 2010, pp.5,10

[13] Takayuki Banzai, Koizumi, H., Kanbayashi, R., Imada, T., Hanawa, T., Sato, M. 2010. *D-Cloud: Design of a Software Testing Environment for Reliable Distributed Systems Using Cloud Computing Technology*, Proceedings of 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010. pp. 631,636

[14] Bhathiya Wickremasinghe, Rodrigo N. Calheiros, Rajkumar Buyya. 2010. "CloudAnalyst: A CloudSim-Based Visual Modeller for Analysing Cloud Computing Environments and Applications", 24th IEEE International Conference on Advanced Information Networking and Applications (AINA), 2010.

[15] Mauel Oriol, Faheem Ullah. 2010. YETI on the Cloud, *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on* , vol., no., pp.434,437, 6-10 April 2010

[16] W.K. Chan, Lijun Mei, Zhenyu Zhang. 2009. Modeling and Testing of Cloud Applications, Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific, pp,111,118

[17] Yang, Yang; Onita, Colin; Dhaliwal, Jasbir; and Zhang, Xihui. 2009. TESTQUAL: conceptualizing software testing as a service, In the 15th Americas conf. on information systems, 6-9.08, San Francisco, California, USA, paper 608, 2009.

[18] Reena Mathew and Ryan Spraetz. 2009. Test Automation on a SaaS Platform, *Proceedings of International Conference on Software Testing Verification and Validation,* 2009, pp.317,325

[19] Stephan Weißleder and Hartmut Lackner. 2013. Top-Down and Bottom-Up Approach for Model-Based Testing of Product Lines. In *Proc. MBT'13*, EPTCS 111, pp.82,94, 2013.

[20] Stelios Sotiriadis, Euripides Petrakis, Stefan Covaci, Paolo Zampognaro, Eleni Georga, Christoph, Thuemmler. 2013a. An architecture for designing Future Internet (FI) applications in sensitive domains: Expressing the Software to data paradigm by utilizing hybrid cloud technology, *13th IEEE International Conference on BioInformatics and BioEngineering (BIBE 2013)*, November 10-13, Chania, Greece, pp.1,6

[21] Stelios Sotiriadis, Nik Bessis, Nick Antonopoulos, Ashiq Anjum, 2013b. SimIC: Designing a new Inter-Cloud Simulation Platform for Integrating Large-scale Resource Management, *27th IEEE International Conference on Advanced Information Networking and Applications (AINA-2013)*, March 25-28, Barcelona, IEEE Computer Society, Washington, DC, USA, pp. 90-97

[22] Michael Unterkalmsteiner, Robert Feldt, Tony Gorschek. 2014 A Taxonomy for Requirements Engineering and Software Test Alignment, *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Volume 23 Issue 2, March 2014, Article No. 16

[23] Alex Galis, and Anastasius Gavras. 2013. The Future Internet: Future Internet Assembly 2013 Validated Results and New Horizons, *Springer Publishing Company*, Incorporated, 2013

[24] Jens Grabowski, Dieter Hogrefe, György Réthy, Ina Schieferdecker, Anthony Wiles, Colin Willcock. 2003. An introduction to the testing and test control notation (TTCN-3), Computer Networks, Volume 42, Issue 3, 21 June 2003, pp.375,403

[25] Ina Schieferdecker, Jens Grabowski, Theofanis Vassiliou-Gioles, George Din. 2008. The Test Technology TTCN-3, Formal Methods and Testing, Lecture Notes in Computer Science, Volume 4949, 2008, pp. 292,319

[26] Wang Jun and Fanpeng Meng. 2011. Software Testing Based on Cloud Computing. In *Proceedings of the 2011 International Conference on Internet Computing and Information Services* (ICICIS '11). IEEE Computer Society, Washington, DC, USA, pp.176,178

[27] C. R. Lima Neto and V. C. Garcia. 2013. Cloud testing framework. In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering* (EASE '13). ACM, New York, NY, USA, pp.252,255

[28] Jerry Gao, Xiaoying Bai, Wei-Tek Tsai, and Tadahiro Uehara. 2013a. Testing as a Service (TaaS) on Clouds. In *Proceedings of the 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering* (SOSE '13). IEEE Computer Society, Washington, DC, USA, pp.212,223. DOI=10.1109/SOSE.2013.66

[29] Koray Incki, Ismail Ari, and Hasan Sozer. 2012. A Survey of Software Testing in the Cloud. In *Proceedings of the 2012 IEEE Sixth International Conference on Software Security and Reliability Companion* (SERE-C '12). IEEE Computer Society, Washington, DC, USA, 18-23

[30] Jerry Gao, Xiaoying Bai, W. T. Tsai, and Tadahiro Uehara. 2013b. SaaS Testing on Clouds - Issues, Challenges and Needs. In *Proceedings of the 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering* (SOSE '13). IEEE Computer Society, Washington, DC, USA, pp.409,415

[31] Ashfaque Ahmed. 2014. *Software Testing & Quality Assurance: From Traditional to Cloud Computing Learn Software Testing & Quality Assurance from the Expert with 25 Years of Experience* (1 ed.). CreateSpace Independent Publishing Platform, USA

[32] Wei-Tek Tsai, Peide Zhong, Janaka Balasooriya, Yinong Chen, Xiaoying Bai, and Jay Elston. 2011. An Approach for Service Composition and Testing for Cloud Computing. In *Proceedings of the 2011 Tenth International Symposium on Autonomous Decentralized Systems* (ISADS '11). IEEE Computer Society, Washington, DC, USA, pp.631,636

[33] Sotiriadis, Stelios, Zampognaro, Paolo. 2013. FI-STAR Back End Validation and Test, available online: https://bscw.fi-star.eu/pub/bscw.cgi/d65098/D2.4-FI-STAR%20Back-End%20Validation%20%26%20Test_v1.0.pdf

[34] Sotiriadis, S., Zampognaro, P., Petrakis, E. and Bessis, N. (2015) "Automatic Deployment of Cloud Services for Healthcare Application Development in FI-STAR FP7", The 30th IEEE International Conference on Advanced Information Networking and Applications (AINA-2016), Le Régent Congress Centre, Crans-Montana, Switzerland, March 23-25, 2016

[35] Sotiriadis, S., Bessis, N., Anjum, A., Buyya, R., "An Inter-Cloud Meta-Scheduling (ICMS) Simulation Framework: Architecture and Evaluation," IEEE Transactions on Services Computing, vol.PP, no.99, pp.1,1doi: 10.1109/TSC.2015.2399312

[36] Vakanas, L., Sotiriadis, S. and Petrakis, E. (2015) Implementing the Cloud Software to Data approach for OpenStack environments, Adaptive Resource Management and Scheduling for Cloud Computing, Held in conjunction with PODC-2015, Donostia-San Sebastián, Spain, on July 20th, 2015