Testing Software Services in Cloud Ecosystems

Mariam Kiran, School of Electrical Engineering and Computer Science,
University of Bradford, Bradford, UK,
m.kiran@bradford.ac.uk

Anthony J H Simons, Department of Computer Science,
University of Sheffield, Sheffield UK,
a.j.simons@sheffield.ac.uk

***Abstract*—**Testing in the Cloud is far more challenging than testing individual software services. A multitude of factors affect testing, including variations across platforms and infrastructure. Architectural issues include differences between private, public Clouds, multi-Clouds and Cloud-bursting. Platform issues include cross-vendor incompatibility, and diverse locales of service deployment and consumption. Software issues include integration with third-party services, the desire to validate competing service offerings to similar standards and need to re-validate services at different stages of service lifecycle. A complete approach to testing whole Cloud ecosystems should involve all relevant stakeholders, such as service provider, consumer and broker. When testing Clouds, the methodologies used should not hinder the advantages Cloud usage brings to the users or programmers and more importantly be simple and cost effective. However, these testing methodologies differ according to the various kinds of Cloud ecosystems and the different user perspectives of the actors involved such as the end-user, the infrastructures, or the different software (i.e. web services). This paper also studies the state-of-the-art in Cloud testing where most research focuses predominantly on web services, functional testing and quality-of-service, usually being considered separately. We suggest a framework, Quality-as-a-Service (QaaS) which integrates quality issues such as functional behaviour and performance monitoring with lifecycle governance and security of the service. This paper maps out the themes in the contemporary research literature and links them with the service lifecycle process for validating future Cloud services. Along the way, we identify important research questions that the future Cloud service testing agenda should seek to address.

Keywords— Web application testing, Cloud computing, software testing, functional and non-functional requirements

1. Introduction

Cloud computing is regarded as a new software delivery paradigm and a 5th utility service after water, electricity, gas and telephony [30]. Businesses are shifting their technologies to the Cloud in order to save on the costs of infrastructure, maintenance and personnel. However there are various risks associated with using the Cloud and new research is turning to building trust and security standards in order for the customers to use the Cloud with greater confidence [34, 35, 36]. Testing becomes an essential part of these standards.

Software testing is challenging and expensive requiring time and resources to scrutinize the application's reliability, functionality and performance. Testing for the Cloud should, in essence, be simple and cost effective; and go beyond the traditional kind of functional and non-functional testing practiced by developers. This is because there is still a need to build trust within the community of Cloud users, and the need for repeated revalidation of the same software, as it is extended, customized or migrated from one platform to another. The specific testing approaches used differ depending on the various kinds of Cloud ecosystem, which vary according to infrastructure, platform and software architecture and the different stakeholder roles involved. Testing should cover the functional and non-functional aspects of the services. In this paper, various testing issues are discussed in the context of future Cloud ecosystems, which will require a spread of testing methods to validate services at the different interaction points between the stakeholders. The paper also surveys the current state-of-the-art in Cloud testing, identifying the gaps between this and the needs that any future complete Cloud testing methodology should satisfy.

Figures 1 and 2 describe various testing dimensions across Cloud platforms. Depending on the different kinds of platforms, depicting the Cloud environment being used, testing issues vary across these. Examples of these issues include functional testing methods, penetration testing or multi-tenancy testing. These depend on the scenario to show which ones would be relevant to the Cloud ecosystems being used. For instance security testing may have a greater influence in multi-cloud environments rather than private Cloud environments. Besides the environments, Clouds exist in three forms depending on the functionality being offered:

- SaaS (Software as a Service): Uses the Web to deliver third-party applications to Clients. For example:Gmail;
- PaaS (Platform as a Service): provides framework to build applications on top as well. It provides the

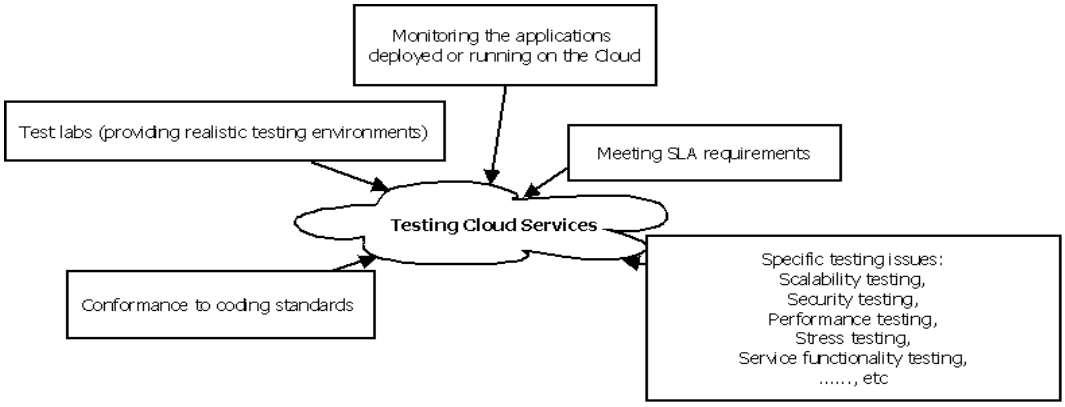

Fig. 1.Examples of testing issues for various Cloud aspects

computer infrastructures, hardware and highly scalable. Example: GoogleAppEngine [50], Heroku [51];

- IaaS (Infrastructure as a Service) third party allows you to install a virtual server on their IT infrastructure.

Testing SOA applications, where most literature exists, focuses on the SaaS functionality of Clouds. Further work for the other two functionalities may need to include a collection of testing issues as shown in Figure 2. These are discussed below in this section with an extensive discussion on related issues of testing with Cloud-related aspects, highlighting, functional, non-functional, service oriented architectures and specific Cloud issues.

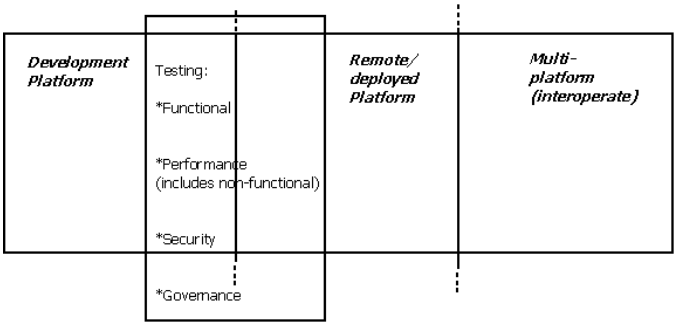

Fig. 2. Cloud Testing issues lie across the platforms being used.

This paper aims to review and assess the current body of knowledge related to Cloud testing, identifying new research directions. The paper is organized into six sections. Following the introduction, section 2 describes the background and the current research trends in the field of Cloud computing testing. The section also discusses in detail the testing scopes and tools presently available, presenting techniques for functional and non-functional testing requirements. Section 3 describes the stakeholders involved and how the stages in the different Cloud ecosystems can be tested. Section 4 presents the future research directions that testing should have in Cloud ecosystems towards Quality-as-a-service, with the methodology presented in Section 5. Finally, Section 6 summarises the argument with the problems which may still persist on a larger context and should be considered in future research issues.

2. Background

A. Current Research Trends

In order to review the vast literature in Cloud testing, various resources were consulted to help classify the published articles according to the sub categories in which most work has been done. Various research databases were queried such as IET digital library, Science Direct, Scopus and Web of Science to produce Figure 3 to show the directions in which most current research trends are focused.

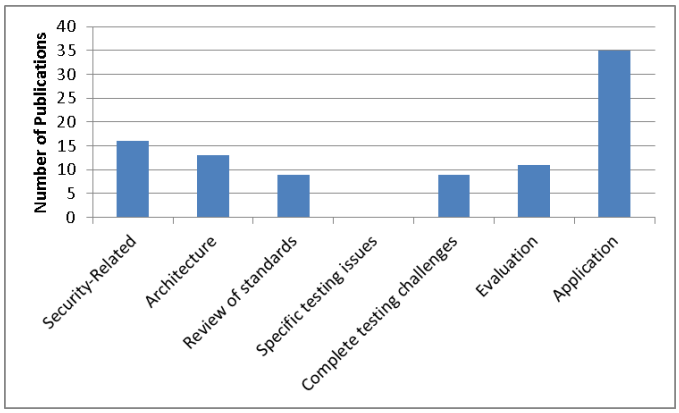

Fig. 3. Categories of publications when searching for "testing Cloud computing" papers (2004-2014).

Figure 3 shows that although most research is focused on the applications of Cloud computing, security issues are more researched than architectural and testing issues. Most of the testing publications identified discuss the challenges in the field but do not focus on any specific type of testing for the Cloud. This could be because this is still an open research question, where researchers are focusing their efforts in identifying the issues present and will work on specific answers to these issues in the future.

Researchers have highlighted specific testing research issues that can be investigated individually. The specific Cloud testing themes have been summarized in Table 1. Katherine et al. [37] discuss various risks associated with

cloud testing such as security, the lack of standards and its usage. However in Table 1, the authors fail to go into details of the attributes tested with each method and the risks involved. Also all the testing methods mentioned only cover non-functional testing, not including functional attributes. Table 1 describes some of the testing targets to be achieved. However following is list of the testing scope categorized based on [26, 5, 37]:

- *Unit testing* - Verifies the modules of a system in isolation before deploying them. Can use either white-box testing, based upon the assumption that knowledge about the system being tested is available or black-box testing, using the output pages produced and compares them with expected results according to the requirements [31]. For stateful and stateless services, unit testing can be used differently. For stateless services, each operation of a service is treated as a unit. However, when testing stateful services, the sequences of operations need to be considered. Examples include WSDLTest [23] a tool to generate random SOAP requests for services expressed in WSDL and SOAP syntax [11]. Unit testing covers the functional requirements but can be costly where the test cases are generated manually. Since the service-under-test may collaborate with other services, testing may either be conducted using a context of live web services or simulated stub or mock services as the context.
- *Interoperability testing* - Refers to the ability to check that multiple components can work together. Kumar et al. [10] described the interoperability issues of core web service specifications such as SOAP, WSDL and UDDI and explained how the WS-I Basic profile provided solutions for the interoperability issues with web service specifications. Yu et al. [28] proposed an ontology-based interoperability testing approach using communication data among web services storing it in an ontology library. This data is later analysed with rules for error analysis with a reasoning framework.
- *Collaborative software testing* - Refers to testing where multiple stakeholders are involved in a web service, such as the developer, integrator, tester and user, all participate in the testing process. It is used in testing techniques such as a usability walk-through, where correct functionality is tested with the participation of different stakeholders. Bai et al. [2] use contracts to allow the test provider to supply specification-based test case designs for the other participants. Testers can run synchronised tests and publish results on services.
- *Integration testing* - Is needed because SOA allows multiple loosely coupled and interoperable distributed services to form a software system. By performing integration testing, all the elements of it can be tested including services, messages, interfaces, and the overall composition. This includes testing of services at binding phase, all workflows and business process connectivity. Yu et al. [29] addressed interaction problems within OWL-S compositions among participating web services using interaction requirements. Integration is different from interoperability as it focuses on services coupled together to form a system, whereas interoperability focuses on if systems can work together.
- *Regression testing* - Performs all the functional tests again, after some change to the integration, reusing the existing test cases from the previous system tests. Regression testing is only about re-testing a component, subsystem or system, after changes have been made, in order to detect regression. Ruth et al. [21] propose a regression testing approach that assumes that the CFGs of services are provided by developers.

The right vendor for the infrastructure needs to be chosen which will be able to mimic the requirements in the SLA, satisfy hardware, software and legal requirements. Sometimes the applications are too tightly coupled that it raises issues in terms of the complexities and the binding contracts. Security and trust are also important, when the applications harbour confidential data, becoming a top priority for users.

TABLE I. TESTING SPECIFIC TO THE CLOUD. C.F. [13, 14, 47].

Testing Approach	Description
System integration testing	Verify that the cloud solution will work within the current infrastructure environments. This will prove implementation of a cloud solution will not impact existing systems.
User acceptance testing	Verify the current cloud solution from the vendor, meets the business needs of the organization of the infrastructure.
Security testing	Ensures that all sensitive information stored in the cloud will be secure.
Performance testing	Measures the system performances in cloud to determine throughput and capacity statistics of the back end service across a range of input and client variance for verifying service level agreements. This also identifies bottle necks, potential weaknesses and performance dependencies. Examples include verifying the network latency, response time, load balancing, peak request count by hosting subscription in different data center across the globe. Traditional load stress testing required validating business scenarios in cloud models for varying the dynamic load and stress on the application.
Disaster recovery testing	Verify the time it takes to recover such as if the system crashes under high load/volume of data, hardware failures, system failures, network outrage, insufficient bandwidth, as per defined in the SLA. Also verify is there any data loss in this process and time it takes to report the failure.
Availability testing	Cloud offering should be available all the time for the enterprise or end user. This would be one of the key responsibilities of the provider to maintain as per the SLA.
Scalability testing	Ensures cloud providers are offering scale in and out functionality as per the demand from the user/organization.
Multi-tenancy testing	For multiple tenants, the concept of multi-tenancy is to provide solution/offering from a single instance to multiple user/clients (tenants). Cloud offering should be validated in terms of security and data not being compromised.
Interoperability testing	Although not particularly testing, this can measure whether the service design or characteristics comply with standards and best practices of the providers. Verifies moving application from one cloud to an alternate cloud provider should have flexibility to run successfully. There should be no issues if the business/user is migrating from one infrastructure to another.
Accessibility testing	Verify user groups across different geographic location are accessible to the cloud at any time without any delay,

	keeping in alliance with the SLA requirements of host locations.
Automation testing	Ensure that the automation suite can be created and executed with minimal changes in the cloud.
Functional testing	Provides the ability to verify the behaviour of the service against a specification of its expected behaviour, builds test suites to assess this.
Security testing	A kind of testing, particularly "penetration testing", which seeks to get past security protocols. Security as a whole involves static design issues, as well as run-time verification of security, security is a measure of reliability to test if the data is secure assessing in terms of vulnerability, availability and integrity.

B. Specific Available Testing Tools

Cloud computing may also adopt three broad styles of software architecture, when communicating between nodes. The oldest style uses straightforward HTTP requests and responses, known as Representational State Transfer (REST). This is a "lightweight" approach, where the client is a simple web-browser and data is transferred in compact HTTP formats; but it requires bespoke server-side processing to dispatch requests and does not necessarily lead to homogeneous systems.

A second style reuses concepts from Service-Oriented Architecture (SOA), a mainstay of traditional web-services. SOA adopts "heavyweight" XML standards, using SOAP for message communication, WSDL and UDDI for service description and discovery. SOA technology supports an open, extensible, federated and composable architecture. SOA fosters the separate development of autonomous, modular software components, which can be reconfigured later in various ways before usage [3]. In this respect, SOA is vendor-diverse, offering the prospect of reusable, interoperable web-services [1]. SOA also offers means of describing and testing Quality-of-Service (QoS). Testing SOA applications is complicated because of their distributed nature. Plentiful research has been undertaken in this direction [7], where programmers have demanded a more centralised approach for managing testing [15].

Whereas both of the above styles depend on the coarse-grained HTTP request and response cycle, a third and increasingly popular style develops bespoke rich-client desktops, providing App-like services that use continuous information trickle via AJAX to communicate with back-end servers. Rich-client applications are developed in client-side scripting languages, such as JavaScript, resulting in "thick client" MVC applications. This architecture presents a completely different set of testing challenges [32, 33] and like RESTful services, does not lead to homogeneity.

Much research has been conducted, for developing tools, to test SOA, which arguably may also apply to the Cloud [5]; however there is also some research on `Testing as a service' (TaaS) for the Cloud [37, 39]. This allows an application to be tested online before deploying it, taking advantage of the benefit of the Cloud by outsourcing the issue. Vengattaraman et al. [38] used modelling tools to focus on the relationships between the applications and the services being tested but lacks the intricate details of how these will be done. TaaS can be presented as two views, which focus on service testing from the viewpoints of the developer and the end-user [40].

Other examples of commercial tools include OASTA CloudTest [41] for performance testing of Web applications, which can simulate thousands of virtual users visiting a website simultaneously, using either private or public cloud infrastructure service. iTKO LISA [42] aims to provide a cloud based environment and virtual services for composite application development, verification and validation supporting continuous integration for development and testing. Another example, Cloud Testing [40] supports cross browser and functional testing of Web applications. Banzai et al. [43] developed D-Cloud, as a dedicated simulated test environment built upon Eucalyptus, using open-source virtual machine software to build a virtual machine for simulating faults in hardware including disk, network and memory. Parveen et al. [44] used a JUnit test framework on the Hadoop platform where the function received the test jobs as experiments to run. Ciortea et al. [45] introduced Cloud9, a cloud based testing service that promised to make high quality testing fast to run on large shared clusters of computers harnessing the aggregate memory and CPU resources based on utilities like Amazon EC2. However these tools show that each of them can be grouped under certain categories, but they need to be merged together to form a complete testing methodlogy for Clouds during the complete service lifecycle.

Most SOA testing is focused on unit testing, for example, just the messages being communicated. However, SOA deployments are complex in terms of WSDL schemas and message patterns. Mostly unit testing focuses on the simple request-response testing of the service's functionality, measuring the correctness of its behaviour. Sometimes these unit behaviours depend on other external business functional units that should also be considered. In some cases, performance testing depicts how non-functional attributes can be tested, which involves a verifying the QoS and could be conducted offline, or in sandbox environments using actual traffic patterns. Additionally, interoperability testing involves a run-time assessment and handling of message patterns that fall out of the expected patterns. Security may also have various issues with injection attacks. The trust perspective with security is also an important emerging standard of W3C which is difficult to define. Bozkurt et al. [5] present a survey of techniques and approaches that have been proposed for testing web services. Most of these techniques have high costs in terms of the test case numbers generated and also cover only some of the facets of SOA. Issues like testing QoS, security, trust or complete system testing still lack established research standards.

Various functional testing tools for web applications exist, such as LogiTest, Maxq, Badboy and Selenium, which are based on capture and replay facilities – recording the interactions that a user has with the graphical interface and repeats them during regression testing. Another approach to functional testing is based on HttpUnit, which is a Java API providing all the building blocks necessary to emulate the behavior of a browser trace, event sequence and form comparison. Marchetto et al. [32] discusses the Document Object Model (DOM) of the page manipulated by Ajax code abstracted into a state model. The callback executions triggered by asynchronous messages received from the web

server are associated with state transitions. Test cases are derived from the state model based on the notion of semantically interacting events. Mesbah et al. [33] discuss invariance based testing in web services such as DOM validity, error messages, discoverability, back-button compatibility and the DOM-tree invariants that can serve as oracles to detect such faults. Generic invariant checking components can be used with a plugin-mechanism to add application-specific state validators and generation of a test suite covering the paths obtained during crawling. Both [32] and [33] use rich-client applications using AJAX and Selenium tests to drive the DOM tree through its states.

Poor service observability and a lack of control over the infrastructure resources are some of the issues cited when designing specific testing methods for the Cloud. The cost of testing services is sometimes high, due to service disruptions and the effects this may have on the system.

C. Testing Functional and Non-functional Requirements

Due to the complexity of Cloud ecosystems, testing can involve a number of functional and non-functional attributes, which can influence the performance of the services when they go live. Morris et al. [16] has highlights various attributes that need to be tested such as functional behaviour of basic operations, and non-functional attributes such as availability of the service during different times of the day.

Functional requirements specify what the system should do, in terms of the specific behaviour of its functions, but may also include ancillary certification requirements defined in the SLA. Examples include rules of business (specific behaviour defined in SLA), authentication methods, certification requirements, historical data or legal requirements. Examples of functional testing methods include:

- *Model-based testing:* Builds a model for the target applications where a complete model exists to serve as the oracle, which describes the desired behaviour [31, 5, 6]. Model-based verification symbolically checks that the model is internally consistent. Examples include using graph or path algorithms for defining the flow of the behaviour [6, 8] or using finite state models where stream X-machines can be used to model data and control of the system [20, 31]. The test case generation for web services can be useful to allow a complete state based testing of the service. *Specification-based test case generation* is similar to model based testing, where the model is more formal. Examples include using a graph transformation to model rules on individual web services as preconditions and updates to the world state [9] or a registry-based testing approach, where the provider augmented the WSDL documents with behavioural descriptions written in UML to generate a complete set of test cases for validating behavioural conformance [5] or using descriptions of service operations (Input, Output, Precondition and Effect) using Semantic Web Rule Language rules and OWL ontologies to derive an extended finite state machine [22]. Ramollari et al. [20] used the Rule Interchange Format – Production Rule Dialect using explicit state updates to working memory to derive a Stream X-machine from which exhaustive tests could be generated [31].

- *Contract-based test case generation*: is also a kind of model based testing that focuses on testing single operations in isolation to validate their pre and post conditions. It defines preconditions under which an operation may be legally accessed, and the post conditions that assure the operation succeeded upon completion. Example include a black-box testing approach for services, by including paired input-output dependencies, invocation sequences, hierarchical function descriptions and sequence specifications [24].

- *Partition testing or equivalence partition testing* is also a kind of model based testing. It develops a strategy for selecting inputs for system functions and observing outputs that indicate that each different categorical response of the system was correctly triggered. This technique aims to partition the input domain into each significant sub-domain, generating test cases for each. Examples include using a category partitioning method with XML schemas for XML based partition testing [4] or using Δ-grammars and WSDL definitions to test the evolutions of services [8]. Sometimes this approach is supplemented by mutation testing, where faults are introduced in every sub-domain to measure the effectiveness of the test suites. Here the specification is used to suggest inputs that trigger different responses.

Non-functional requirements specify how a system should perform, in terms of its efficiency and reliability. Some of these aspects can also be defined in the SLA, such as response time, scalability, reliability, availability, security or maintainability. Testing examples, are performance testing, security testing or dependability testing for satisfying customer needs [46]. Examples of methods include:

- Other testing issues include using already present standards [26, 19, 27]. The XML standard can be validated using standard XML compliance tools based on DTD or schemas [25]. WSDL descriptions must conform to some published ontology, that can be checked using profile tools. SOAP messages can be tested for compliance to message protocols using WS-I. A popular language supporting full behavioural description of services is BPEL or WS-BPEL (Web Services Business Process Execution Language) [17]. It is commonly used as a way of describing workflows as a flowchart, although it also supports other procedural and state-based views. It has become the glue for orchestrating services in many SOA applications. However, testing from BPEL only offers little abstraction as a model for testing.

- Testing Service-centric Systems for QoS Violations: QoS ratings must be published by the provider and be accessible to consumers within the SOA environment. The importance of QoS in Web services and problems surrounding have led to service standards called WS-Agreement, a specification language for standardising

the overall agreement structure [49]. The QoS testing cost is high due to the cost of service invocation and the need to generate test data that simulates real usage. Multiple test case executions are needed to provide the average of the results from the runs for a realistic QoS score, rather than a single test. These QoS scores are also updated periodically from the monitoring data.

- *Fault-based testing*: A specific approach that tests the robustness of a system, usually by fault injection into the code. It presupposes that other tests already exist to detect the faults (not a test generation method, but a test quality). XML/SOAP perturbation uses faulty messages in SOAP from the captured messages and injects faults before sending those where network fault injection involve corrupting, dropping or reordering the network packets [12]. Mutation of web services for detecting errors defines mutated operators in contracts checking for semantic adequacy in web services [15].
- *Testing semantic behaviour*: A web service can be tested in conformance with key standards like validating a service interface using WSDL. Web service authentication is verified using WS-Security, WS-Digest and non-proprietary web services specification such as SOAP, WSDL and UDDI [16]. Validating the service binding and messaging in conformance with SOAP protocol over HTTP involves checking request-response message pattern, response message exchange pattern, action feature and the bindings. Validating web service interoperability involves using standards such as WS-I Basic Profile 1.0, Simple SOAP Binding Profile 1.0, SOAP messages with attachments, standard UDDI version 2.0 or XML Schema standards. For web service compositions, testing for conformance is more problematic. WS-BPEL is gaining widespread acceptance for web service orchestrations, with several tools to parse it and flag non-conforming use. However, this can be built without a conformance check which makes this difficult to be validated [18].

3. Testing Methodology in Cloud Scenarios

Stakeholders

Testing for Clouds is economically important, in the sense that it will protect the investment of businesses that rely increasingly on the Cloud. Along with the purely technical challenges of testing the integration of infrastructure, platform and software services with legacy systems, the social environment in Cloud Computing comprises a number of roles which play a part during service certification. The NIST Reference architecture [49] defines five main stakeholder roles: (i) the Cloud Consumer, who uses services, (ii) the Cloud Provider, who provides services, (iii) the Cloud Auditor, who independently assesses the security issues, (iv) the Cloud Broker, who acts as an intermediary between providers and consumers and (v) the Cloud Carrier, who provides the network connectivity and transport of services.

The heterogeneous nature of Clouds involve many different kinds of stakeholder, who integrate many packages operating asynchronously. Table 2 presents a compilation of the various stakeholders involved with different testing issues.

TABLE II. STAKEHOLDERS AND TESTING FOR FUNCTIONAL AND NON FUNCTIONAL REQUIREMENTS.

Stake-holder	Actions performed	Testing issues involved and re-quirements addressed
Service developer /Programmer	Create a service from scratch, reuse an available service. Or create an interface for services involved by using an existing component and wrapping it to perform as a service.	Construct a test suite or guidelines for testing the independent/wrapped service before deploying. Prefer to test the service locally. Tests functional requirements such as performance and SLA defined variables.
(Enduser) Customer	Uses the services, asking for it in the form of an SLA request. Can also use applications that employ a service.	None. Just requesting services..
Service provider	Provides services as a form of SLA. Develops the various governance processes that support the service agreement and service consumers.	Establishes customer support issues to satisfy defined in the SLA. Both functional and non-functional requirements have to be met for business satisfaction.
Service consumer	Executes services.	Writing tests and standards necessary to achieve assurance about service performance. Both functional/non-functional requirements ensure SLAs.
Service integrator	Uses existing services to create composite services or create an end-user application.	Develop guidelines for testing composites of various types employing range of composition mechanisms expected. Non-functional requirements to ensure integration of services are successful not affecting results.
Infrastructure provider / Platform provider	Provides necessary infrastructure and middleware mechanisms such as service discovery, service providers, service consumers and service integrators. Provides the necessary platform for the services to execute.	Develop guidelines and governance processes for testing and verification of new and revised infrastructure capabilities. Includes notification for users of infrastructure and triggers for retesting. Develop policies for the level of testing support provided by the infrastructure provider to the service provider.
Third-party service tester or certifier	Validates and potentially certifies whether a service works as expected.	Identify focus, expectations and limitations of third-party testing and certification activities. Both functional and non-functional requirements to ensure SLA criteria.
Broker	Acts as an intermediate between the service consumers and providers. Ensures all requirements of services are met and delivered on time.	Can have its own testing methodologies to follow for both functional and non functional requirements for the SLAs to be satisfied.

4. Proposed Steps towards testing during Service lifecycle

There are various steps involved in Cloud ecosystems, which focus on testing issues,

1. Functional Testing during on-boarding: The providers or broker may test services before accepting to execute it.
2. Pre-live sandbox functional testing: The service could be executed in 'safe' or sandbox environment to replicate how it would behave on real infrastructures.
3. Monitoring sandbox performance: To monitor service performance in safe sandbox environments to monitor the QoS with respect to SLA. This covers part of non-functional requirements.
4. Live testing: Execute service on real infrastructures.
5. Monitoring performance: Monitor service performance in the real environment.
6. Penalties issued: If requirements are not met, penalties may be issued to the providers.
7. Reporting on performance and functional testing results: Reporting on final results of the service and it is performance to the providers and end users.

5. Testing in different Cloud Ecosystems

Figure 4 describes the various Cloud ecosystems and the stages at which they can incorporate the steps (1-7) to test the different aspects during the service lifecycle. In the private cloud scenario, the infrastructure or platform provider can test the service before on-boarding it. It can then run and be tested in a sandbox, or in the real environment, and report the results back to the end-user. Similar processes happen in the Cloud bursting case, where one of the infrastructure providers shares the service load with another provider. Each provider would then run and test independently.

In the multi-cloud scenario, the service provider/end-user communicates to a number of infrastructure providers, which independently test their service executions. During the brokerage scenario, the broker acts as an intermediary, taking most of the responsibility of testing and collecting the results.

There are various interactions taking place between the providers, brokers and the customers during the service lifecycle. These have been explained as follows:

- Provider registration for brokers: Infrastructure and platform providers (and service consumers) register with brokers with their characteristics and policies. This includes certifications, testing methodologies and capabilities available for services. They may also agree on what data will be shared with brokers during service operation in case it is against company policies.
- Service on-boarding: The service provider sends an offer of a service provision to brokers or infrastructure providers. The Broker/infrastructure provider then inspects the declared functional behaviour of the service and inspects its declared performance characteristics for example time to execute. It tests the service on particular platforms and infrastructure, to ensure that the service executes correctly and in time. Functional test suites are created from the declared service specification. The broker can then choose a platform or infrastructure on which to deploy the service.
- Service operation: The Broker or the infrastructure provider monitors the pre-live sandbox performance of the services deployed on infrastructure measures the Quality of Service according to declared QoS in the

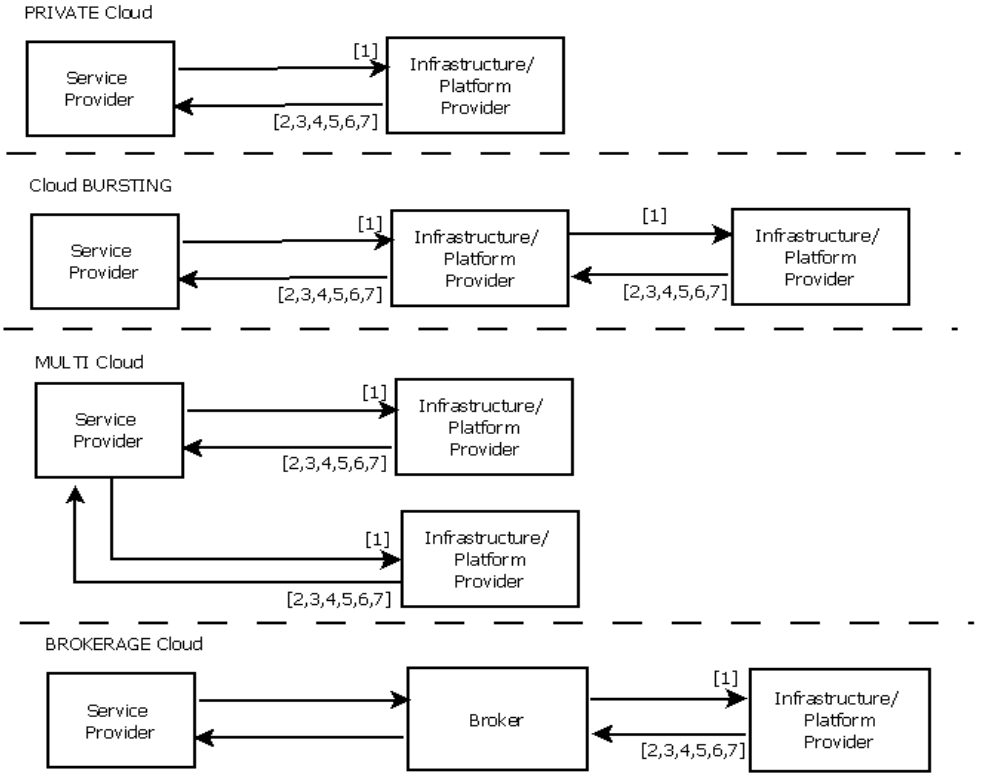

Fig 4. Testing steps in the different Cloud scenarios.

service specification (SLA). If the sandbox performance of the deployed service fails to meet the service's declared SLA, the broker/infrastructure provider notifies the service provider and does not allow it to go live. The QoS can be monitored once it has gone live. If this is about to fall below agreed thresholds, the broker alerts the relevant consumers and provider. In case the SLA is being broken, the broker may impose something like a cost penalty (on the provider) which can also be specified in the SLA. The Broker/infrastructure provider monitors the run-time functional behaviour of the service, using parallel models generated from the specification. Any run-time violation of behaviour is reported back to the service provider and the service consumer. In case the run-time functional behaviour of service is compromised, the service provider must trigger some compensation to revert to a stable state in which all parties are satisfied. Again, this can be specified in the SLA.

6. Towards Quality as a Service

The literature review highlights that testing in Cloud ecosystems is a vast topic. Moving from private clouds to ecosystems, the need for trust among parties will grow, and the need for generally adopted methods and mechanisms for quality control will grow. Automated testing techniques may form a part of this. Having a complete testing methodology which considers all aspects of functional and non-functional requirements is a challenging task. Some issues can be broken down as follows:

- Find an efficient manner for runtime testing with binding of services or multi-tenancy tests for multiple users of the services and the different tools using them.
- The same web service can have different implementations. These can be based on the same specification or be able to cover both perspectives.
- Issues like security and trust need also to be defined in some manner to be tested as well.
- Be cost effective and run multiple tests at same time.
- Can a model be drawn up from a stateless protocol like HTTP, to test its functionality? Services are usually stateless or stateful depending on their business logic. A model can be designed as a mental abstraction of how a service should perform as a starting point.
- A service can be tested various stages – (i) pre-deployment by the provider; (ii) post-deployment on the target platform, with toy loads; and (iii) live during operation, with realistic loads.
- How can integrated service be tested to check they do not break other services when they go live?
- Multiple issues affecting the service such as multiple users handling similar data, or security requirements of the Cloud need to be tested.
- How to test the non-functional requirements such as performance and load?
- When do we know when testing is enough for the complex system depending on the notion of coverage?
- Generating an abstract test suite that converts the high-level suite into actual code or scripts that can exercise his service application.

Automating the process can allow providers to submit valid specification for testing the services, supplied with the SLA. The tools can then decide to use one of the advertised standard specifications for the service supplied. This can then be used as a complete specification generating high-level test suite to some agreed measure of state coverage of the services. This can be executed and any discovered faults noted, fixing the service until the tests pass. The fully-tested service is then used with its generated grounded test suite for regression testing, containing the elements above, offered to customers who want to sandbox-test the service. If the provider modifies the service, this process must be repeated with a revised specification.

An alternative path that may be possible is that the Broker or infrastructure provider may generate one of several *standard* grounded versions of the high-level test suite for a particular kind of service technology, such as SOAP communication or RESTful communication. This is only possible if the service API is built in a standard way that every provider agrees to observe. The provider submits the implemented service and the grounded test-suite on the intended platform for the software, and notifies the provider of any errors. The provider must then fix the service until all tests pass when the Broker or provider executes the software. The service can be tested on a simulated environment or a real environment depending on the costs and risks the business is willing to take. For Cloud applications it would be ideal to have these automated with minimum human interaction, only responding if something has gone wrong.

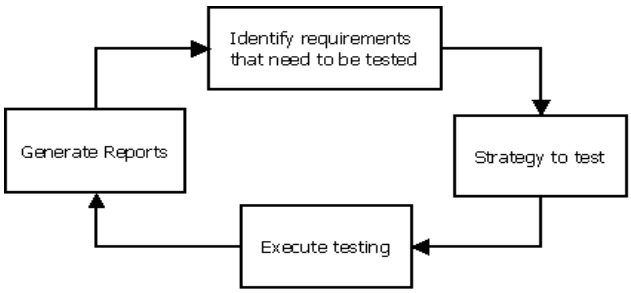

Fig 5. TestCycle: The testing lifecycle.

7. Towards A Methodology

Figure 5 presents a testing cycle which contains the basic functions needed to be carried out for testing. The requirements can be presented by a test library which contains functional and non functional elements that need to be tested. Depending on the requirements, the strategies chosen can be specific, for instance for functional testing a strategy of model-based testing can be used. The results generated in a report

are the output for the testing phase. The TestCycle can exist at all stages of the service lifecycle as shown in Figure 6. The cycle would allow complete testing issues to be covered depending on the stage the service exists in.

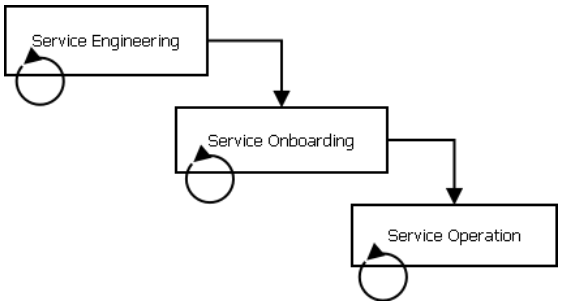

Fig 6. The TestCycle exists at every stage of the service lifecycle.

The various categories of the testing requirements can be selected at the various stages of the service lifecycle using the test library. For instance,

- During Service Engineering phase: The service is selected to only test for the following requirements, (i) behavioural tests for the service, functional requirements are chosen, (ii) SLA requirements are tested – non-functional requirements like the load and performance are under a certain limit.
- During Service Onboarding phase: The service may be tested in a sandbox environment to test for functionality and mimic how it would behave when it goes live, (i) behavioural tests for the service – to test the service behaves the correct way when it goes live, (ii) Company policies are tested – Any rules that need to be followed like documentation standards and legislations will be tested for compliance, and (iii) Performance and stress testing for load and capacity monitoring of the service such that it would not fail when running live with multiple transactions.
- During Service Operation phase: The service is continuously tested while it is running live in the Cloud. This will involve live monitoring on the service performance and further tests include, (i) Company policies are still being followed, (ii) Performance and stress testing of the service and (iii) Other tests desired by users like enforcing security issues.

This process of complete testing during the service evolution phase presents a comprehensive testing methodology to ensure all aspects of services are tested. However there could be certain costs with this kind of testing methodology discussed in the next section.

8. Summary

Some problems of testing include service observability and lack of control on the infrastructure resources, which have not been covered above. Cloud testing can provide key benefits where it is possible to sometimes outsource the testing before the service goes live. However there are certain considerations before moving to Cloud testing which need to be kept in mind,

- Testing services can present costs sometimes such as cause service disruptions during testing and effects of testing on the system, which is another thing to consider when developing testing strategies.
- This can also introduce lower expenditure, as the service is trusted and secure when it goes live.
- There is more consumption of hardware resources to test the services.
- Does not support green computing as more resources are being used for testing.
- Cost models need to be tailored per service to help providers decide the cost of business maintainability and the service itself. Testing services before they go live in 'sandbox' environments can help reduce unpredictable behaviour when the service goes 'live'. Most providers have their own systems to minimise the cost of testing in sandbox environments.

Further future work needs to be carried out in constructing a test library and gathering aspects that should be tested for services and Cloud environments in general.

Cloud providers need to find a balance between the testing resources and the actual product before services are made available to be used by consumers. All these procedures aim to build the trust and reduce risks associated with using Clouds as future technologies.

ACKNOWLEDGMENT

The research leading to these results is funded from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no328392, the Broker@Cloud project (www.broker-cloud.eu).

REFERENCES

[1] Arcitura Education Inc, Service Orientation.com, Dec 2012 Online: http://serviceorientation.com/.

[2] X. Bai, Y. Wang, G. Dai, W.T. Tsai, and Y. Chen, A framework for contract-based collaborative verification and validation of web services. Proc. of 10th Int. Symposium on Component-Based Software Engineering, LNCS, H.W. Schmidt, I. Crnkovic, G.T.Heineman, J.A. Stafford, Eds., vol. 4608, USA: Springer, 2007, pp. 258–273.

[3] M. Bell, 2008, Introduction to Service-Oriented Modeling. Service-Oriented Modeling: Service Analysis, Design, and Architecture. Wiley & Sons. p. 3. ISBN 978-0-470-14111-3.

[4] A. Bertolino, I. Frantzen, A. Polini, and J. Tretmans, Audition of Web Services for Testing Conformance to Open Specified Protocols. Reussner, R., Stafford, J.A., Szyperski, C. (eds.) Architecting Systems with Trustworthy Components. LNCS, vol. 3938, pp. 1–25, 2006.

[5] M. Bozkurt, M. Harman, and Y. Hassoun, Testing & Verification in Service-Oriented Architecture: A Survey, In Software Testing Verification and Reliability, 2009 pg: 1–7, DOI: 10.1002/000, Online: http://www0.cs.ucl.ac.uk/staff/M.Bozkurt/files/pdf/Bozkurt_Harman_Hassoun_Testing_Verification_In_SOA_A_Survey.pdf.

[6] A.T. Endo, A.S. Simao,S.R.S. Souza, and P.S.L. Souza, Web services composition testing: A strategy based on structural testing of parallel programs. Proc. of the Testing: Academic & Industrial Conf. Practice and Research Techniques. IEEE Computer Society, 2008, pp. 3–12.

[7] Gartner 2009, Norton D., Feiman J., MacDonald N., Pezzini M., Natis Y., Sholler D., vander Heiden G., Karamouzis F., Young A., James G.A., Knipp E., Duggan J., Murphy T.E., Valdes R., Blechar M., Driver M., Young G., Vining J., Knox R.E., Feinberg D., Hart T.J., Patrick C., Forsman J., Basso M., Simpson R., Adachi Y., Clark W., King M.J., Hill J.B., Gootzit D., Bradley A.J., Kenney L.F., Stang D.B., Hype Cycle for Application Development, 2009.

[8] R. Heckel, L. Mariani, Automatic Conformance Testing of Web Services. Cerioli, M. (ed.). LNCS, 3442, pp. 34–48. Springer, 2005.

[9] R. Heckel, M. Lohmann, Towards contract-based testing of web services, in Proc. of Int. Workshop on Test and Analysis of Component Based Systems, vol. 116, Barcelona, Spain, pp. 145–156, 2004.

[10] K.S. Kumar, A.S. Das, S. Padmanabhuni, WS-I Basic Profile: A practitioner's view, Proc. of IEEE Int. Conference on Web Services. San Diego, CA, USA: IEEE Computer Society, July 2004, pp. 17–24.

[11] C. Lenz, J. Chimiak-Opoka, R. Breu, Model Driven Testing of SOA–based software, in Proc. of the Workshop on Software Engineering Methods for Service-oriented Architecture, D. Lubke, ed. Hannover, Germany, FG Software Engineering, May 2007, pp. 99–110.

[12] N. Looker, J. Xu, M. Munro, Determining the dependability of service-oriented architectures, Int. Jour. of Simulation and Process Modelling, 3, 26, pp. 88–97, 2007.

[13] J. Macy, SOA Testing Tools & Best Practices Oct 2009, www.soatesting.com.

[14] J. Macy, Limits of Opensource SOA testing tools Jun 2009 www.soatesting.com.

[15] H. Mei, L. Zhang, A framework for testing web services and its supporting tool Proceedings of IEEE Int. on Service-Oriented System Engineering. Beijing, China: IEEE Computer Society, 2005, 199–206.

[16] E. Morris, W. Anderson, S. Bala, D. Carney, J. Morley, P. Place, S. Simanta, Testing in Service-Oriented Environments, technical report, CMU/SEI-2010-TR-011, Online: http://www.sei.cmu.edu/reports/10tr011.pdf.

[17] L. O'Brien, L. Bass, P. Merson, Quality Attributes and Service-Oriented Architectures, Technical Note: CMU/SEI-2005-TN-014, 2005, Online: http://www.sei.cmu.edu/library/reports/abstracts/05tn014.cfm.

[18] Organization for the Advancement of Structured Information Standards (OASIS). Online: http://www.oasis-open.org/home/index.php.

[19] OWL-S: Semantic Markup for Web Services. Online: http://www.w3.org/Submission/OWL-S/.

[20] E. Ramollari, D. Kourtesis, D. Dranidis, A.J.H. Simons, Leveraging semantic web service descriptions for validation by automated functional testing, Proc. 6th Eur. Semantic Web Conf., eds. L Aroyo, P Traverso, LNCS, 5554, Greece: Springer, 2009.

[21] M. Ruth and S. Tu, A safe regression test selection technique for web services. Proc. of Second Int. Conf. on Internet and Web Applications and Services. IEEE Computer Society, May 2007, pp. 47–47.

[22] A. Sinha, A. Paradkar, Model-based Functional Conformance Testing of Web Services Operating on Persistent Data. Proceedings of Workshop on Testing, Analysis and Verification of Web Services and Applications, pp. 17–22, 2006.

[23] H.M. Sneed, S. Huang, WSDLTest - a tool for testing web services, Proc. of Eighth IEEE Int. Symposium on Web Site Evolution. IEEE Computer Society, Sept. 2006, pp. 14–21.

[24] W.T. Tsai, R. Paul, Y. Wang, C. Fan and D. Wang, Extending WSDL to facilitate web services testing, Proc. of 7th IEEE Int. Symposium on High Assurance Systems Engineering, Japan, Oct. 2002, pp. 171–172.

[25] World Wide Web Consortium (2009) Online: http://www.w3.org/.

[26] Web Services Description Language (WSDL 1.1). Online: http://www.w3.org/TR/wsdl.

[27] Web Service Modelling Ontology (WSMO). Online: http://www.wsmo.org/.

[28] Y. Yu, N. Huang, M. Ye, Web services interoperability testing based on ontology. Proc. of Fifth Int. Conf. on Computer and Inf. Technology. IEEE Computer Society, Sept. 2005, pp. 1075–1079.

[29] Y. Yu, N. Huang, Q. Luo, OWL-S based interaction testing of web service-based system, Proc. of Third Int. Conf. on Next Generation Web Services Practices, South Korea: IEEE Computer Society, 2007.

[30] R. Buyya, C.S Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as 5th utility. Future Generation Computer Systems, 25, 599 – 616, 2008.

[31] M. Holcombe, F. Ipate Correct Systems Building a Business Process Solution. Springer Applied Computing Series, 1998.

[32] A. Marchetto, P. Tonella, F. Ricca., State-Based Testing of Ajax Web Applications. Proc. of the Int. Conf. on Software Testing, Verification and Validation. IEEE Computer Society, 121-130, DOI=10.1109/ICST.2008.22, 2008.

[33] A. Mesbah, D. Roest, Invariant-Based Automatic Testing of Modern Web Applications, January/February 2012, vol. 38, no. 1, pp. 35-53.

[34] K. Djemame, B. Barnitzke, M. Corrales, M. Kiran, M. Jiang, D. Armstrong, N. Forgo, I. Nwankwo, Legal issues in clouds: towards a risk inventory, Phil. Trans. of the Royal Society A: Mathematical, Physical and Engineering Sciences, vol 371, Issue 1983, 2013.

[35] A.U. Khan, M. Kiran, M. Oriol, M. Jiang, K. Djemame: Security risks and their management in cloud computing. CloudCom 2012: 121-128.

[36] M. Kiran, M. Jiang, D. Armstrong, K. Djemame, Towards a Service Life Cycle-based Methodology for Risk Assessment in Cloud Computing, Int. Conf. Cloud and Green Computing, Australia,2011.

[37] Katherine, A.V., Alagarsamy K., Software Testing in Cloud Platforms: A survey, Int. Jour. of Computer Applications, 0975-8887, 46, 6, 2012.

[38] P. Vengattaraman, R. Dhavachelvan, R. Baskaran, Model of Cloud Based Application Environment for Software Testing, Int. Journal of Computer Science and Inf. Security, vol. 7, no. 3, 2010.

[39] Y. Yang, C. Onita, J. Dhaliwal, X. Zhang, TESTQUAL: conceptualizing software testing as a service, in 15th Americas conf. on information systems, USA, paper 608, 2009.

[40] Cognizant reports, Taking Testing to the Cloud. March 2011.

[41] SOASTA. [Online]. Available: http://www.SOA STA.com/.

[42] ITKO. [Online]. Available: http://www.itko.com.

[43] T. Banzai, H. Koizumi, R. Kanbayashi, T. Imada, T. Hanawa, M. Sato, D-Cloud: Design of a Software Testing Environment for Reliable Distributed Systems using Cloud Computing Technology, Proc. of 10th IEEE/ACM Int. Conf. on Cluster, Cloud and Grid Computing, 2010.

[44] T. Parveen, S. Tilley, N. Daley. P. Morales, Towards a Distributed Execution Framework for JUnit Test Cases, IEEE Int. Conf. on Software Maintenance, 2009, pp.425 –428.

[45] L. Ciortea, C. Zamfir, S. Bucur, V. Chipounov, G. Candea, Cloud9: A software testing service, 3rd SOSP Workshop on Large Distributed Systems and Middleware, Big Sky, MT, October 2009.

[46] Communities, Functional versus Non-Functional Requirements and Testing,http://communities.vmware.com/servlet/JiveServlet/previewBody/17409-102-2-22494/Functional%20versus%20Non-functional.pdf.

[47] K. Sahoo, Overview of Testing in Cloud, Code Project Article 2013,http://www.codeproject.com/Articles/580167/Overview-of-Testing-in-Cloud.

[48] P. Mell and T. Grance, Effectively and Securely Using the Cloud Computing Paradigm (v0.25), NIST, 2009.

[49] A. Andrieux, K. Czakkowski, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, M. Xu, Web Services Agreement Specification (WS-Agreement), Grid Resource Allocation Agreement Protocol (GRAAP) WG, Open Grid Forum, 2007, http://www.ogf.org/documents/GFD.107.pdf

[50] https://developers.google.com/appengine/

[51] https://www.heroku.com/