Taxonomy and Requirements Rationalization for Infrastructure in Cloud-based Software Testing

Philip Robinson and Carmelo Ragusa
SAP Research
SAP UK Ltd
Belfast, United Kingdom
{firstname}.lastname@sap.com

Abstract—Cloud-based software testing is today predominantly focused on testing services provided in the cloud. Secondly, the properties of the testing process are often highlighted as opposed to the infrastructure. A taxonomy of 5 patterns for testing in the cloud and 7 criteria for effective infrastructure is presented. The practicality and relevance of the taxonomy are demonstrated with an application study in the Platform as a Service (PaaS) domain. This domain has been selected as there are no extensive studies on testing PaaS applications or the infrastructure requirements for supporting such tests.

Index Terms— Cloud infrastructure management, requirements engineering, software testing

I. INTRODUCTION

Cloud-based software applications integrate various pieces of software at different levels of the software stack from various providers and hosts. This change in how software is developed, deployed and operated does not remove the need for a rigorous, mature testing process. However the properties and features of existing cloud infrastructure do not support the requirements for effective software testing. Cloud-based software testing is more than software testing as an online service, although there is great demand and technology push in this direction [1]. Benchmarks for software testing processes are not possible, as this will vary from organization to organization. We suggest that more emphasis needs to be given to how infrastructure is used in testing as a measure of effectiveness and maturity in usage of cloud-based facilities for software testing. In this paper we formalize and describe a taxonomy and 3-staged analysis process towards rationalizing infrastructure requirements for effective cloud-based software testing.

Section II discusses the background to the problem domain of testing cloud-based software applications in more detail, while Section III describes the inputs, activities and outputs of the process. The process has 3 analysis activities: infrastructure effectiveness criteria analysis, test process complexity analysis and requirements rationalization, where the latter uses the analysis information from the first 2. Section IV is an application study that applies the methodology, while Section V concludes the paper.

II. BACKGROUND

A. General Issues in Software Testing

Recent surveys [2] indicate that software testing consumes 30 – 50 % of development resources in organisations. However, rigorous software testing is critical as bugs found after software becomes operational are hard and expensive to fix [3]. In spite of its expense it remains critical. Yet every type of application and usage differs, such that best practices tend to be high-level. Each test case must be customized providing a description of metrics and success conditions. This customization is associated with scripts for setting up the preconditions, generating and retrieving the input data, invoking the operations, checking for the outputs and monitoring the environment at the completion of the test execution.

A further issue is selecting and reserving the appropriate physical and virtual infrastructure elements for the test [4]. This is dependent on the size of the software under test, the test suite, the types of payloads that will be used as inputs and outputs, and the size of system instance that represents the software in productive state. Supporting this selection and reservation on demand is one of the key selling points for cloud computing. Although the cloud model promotes resource sharing amongst customers (i.e. multi-tenancy), testing typically requires exclusive access and an isolated, execution environment. The test suite varies depending on the type of test to be performed: functional, regression, performance, security or fault tolerance. The resource selection has to also take into account conditions that resemble its intended production environment.

Finally, the execution of the tests and gathering of results can be automated but depends on the available scripts and supporting technology. The storage of logs and assurance of access to test data is an advantage and challenge for cloud-based storage services. Testing typically operates on non-sensitive data but the results of testing are private to an organisation, as leakage of bugs is valuable to competitors and malicious hackers. For this reason the proper decommissioning of the infrastructure on completion of testing is critical, along with other challenges for infrastructure management.
B. Cloud Infrastructure and Management

The management of a Cloud Infrastructure is a complex task, which depends on the scope of the infrastructure itself. A first issue to consider is the distributed nature of the infrastructure compared to an in-house cluster, which operates a simpler network configuration. As a second issue it should be considered if the cloud infrastructure is within a single administration domain or spawns across multiple domains, in which case it adds on the network complexity, because of different security and network setups. A third issue to take into account is if the infrastructure is made of clouds based on the same technologies or different ones. In the latter case specific “drivers” would need to be adopted as well as specific network configurations might need to be setup (i.e. network addresses, ports, etc.). A fourth issue relates to the type and level of security in place. Single sign-on, certificates, access lists, etc. increase also the infrastructure complexity. Finally, a fifth issue regards the model of how the infrastructure is operated. If it is an open public platform to use for experimenting, with no guarantees then resources are assigned and managed on a best-effort like base. On the other hand, if it is used to create revenue then customers Service Level Agreements (SLAs) are in place and a high level management system will control the resource allocation, SLAs adherence and breaches, corrective actions, etc. This further increases the complexity of the cloud infrastructure to be managed, as well as the overall testing process of applications that are developed or deployed in the cloud.

C. Responsibilities in Testing Cloud-based Software

The interpretation of testing cloud-based software changes based on the properties of the software, the cloud model and the testing resources used by an organization. From our practical experience there are 7 different responsibilities (R1 – R7) involved in any approach to cloud-based testing:

R1-Infrastructure Management: procurement, maintenance and provisioning of physical and virtual servers and network.

R2-Infrastructure Service Management: creation and maintenance of accounts and access to virtual resource instances (e.g. machines, storage).

R3-Software under Test (SuT) development: design, implementation and integration of the software to be tested.

R4-SuT Management: deployment, configuration and interconnection of the software to be tested.

R5-SuT Service Management: maintenance of access to software and creation of instances to be tested.

R6-Test Suite Management: deployment and maintenance of testing software and harness for executing test case.

R7-Test Case Management: management of scripts for test cases and configuration of the testing harness.

These responsibilities are distributed differently across customers and providers depending on the cloud usage model employed. In the context of software testing we refer to 4 cloud usage models:

1. Infrastructure as a Service (IaaS): the service provided is access to compute and storage resources for deploying software (SuT and/or testing suite).

2. Platform as a Service (PaaS): the service provided is a platform for developing and running software (the SuT). Testing can be a value-added offering of PaaS providers.

3. Software as a Service (SaaS): the service provided is pre-built and pre-deployed software, which becomes the SuT.

4. Software Testing as a Service (STaaS): testing is the service provided. This can include consultancy support in establishing test cases but mostly refers to providing a pre-built, online facility for software testing.

Table 1 shows how the software testing responsibilities R1 to R7 are distributed across the roles of customer and provider according to the cloud usage model.

<table>
<thead>
<tr>
<th>Cloud Usage Model</th>
<th>Customer Responsibilities</th>
<th>Provider Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>STaaS</td>
<td>(R1), (R2), (R3), (R4), R5,_, _, [R7]</td>
<td>R6*, [R7]</td>
</tr>
<tr>
<td>SaaS</td>
<td>_, _, _, _, R5, (R6), [R7]</td>
<td>R1, R2, R3*, R4, _, (R6), [R7]</td>
</tr>
<tr>
<td>PaaS</td>
<td>_, _, {R3}, _, {R5}, R6, R7</td>
<td>R1, R2, [R3], R4*, (R5), (R6), _</td>
</tr>
<tr>
<td>IaaS</td>
<td>_, [R2], R3, R4, R5, R6, R7</td>
<td>R1*, [R2], _, _, _, _, _</td>
</tr>
</tbody>
</table>

Table 1. Distribution of responsibilities R1 – R7 in cloud-based testing across a resource/service customer and a provider. Rx* => provider’s main responsibility; (Rx) => optional responsibility; (Rx) => shared responsibility.

The customer has most responsibility for software testing in the IaaS model, as the infrastructure provider plays a passive role that is agnostic to the testing process. The customer also has significant responsibility in the STaaS model, as the role of the provider is only to supply the testing framework. The responsibility of the customer is significantly reduced in the SaaS and PaaS models as the level of software and infrastructure ownership is reduced.

There are however some new challenges for testing cloud-based software that arise in any of these usage models. Firstly, the notion of a “system under test” becomes more complex, given the distribution of functionality and the loss in clarity of the line between development and operation. Secondly, testing
involves multiple organizations and resources from different sources, which brings coordination overhead. For example, consider that regression testing must account for changes in infrastructure and platforms at different providers. Thirdly, isolation becomes a more critical property not only for separating customers, but for separating test and production systems. Consider that a failed test could potentially cause production systems to degrade in performance if no robust isolation mechanisms are in place. Cloud-based infrastructure for test purposes increases the likelihood of introducing vulnerabilities that can be exploited and lead to compromise critical systems.

D. Related work

The main efforts in the area of testing relative to Cloud computing are to use the Cloud as a platform to support and enhance a testing process for traditional (e.g., non-virtualized) applications. In particular, some approaches are in the area of cloud for testing, which use the cloud to run the test load and test suites in order to simulate multiple distributed users avoiding using in-house resources. Several companies offer these type of services, such as STaaS – Software Testing as a Service from Sogeti [5], Infrastructure Optimisation Services - IBM Smart Business Test Cloud from IBM [6], Sauce on Demand [7], which allows testing web applications using Selenium across multiple web browsers in the cloud. Testing Anywhere [8], Jmeter in the cloud [9], just to mention some. On the other hand, research in this area is limited. YETI, the York Extensible Testing Infrastructure, which is an automated random testing tool for different programming languages, has been ported into the cloud [10]. However, although the cloud for testing approach solves an important aspect of the testing phase, it does not look at the issues the infrastructure faces to provide such service.

Other approaches, in the area of testing on cloud, use the cloud to run the application under test and perform functional and performance tests. Cloud9 is an online testing service which parallelizes symbolic executions to automate the testing process and it is able to scale to large clusters. Cloud9 uses Amazon EC2 as cloud platform [11]. In [12] and [13] the authors automated the software testing as a service process reducing human intervention, claiming also economic benefits. D-Cloud is an infrastructure to test dependable parallel and distributed systems specialized in fault-tolerance tests [14].

Other approaches run both load and application in the cloud. However, none try to study the impact of testing applications in the cloud from the infrastructure perspective. Also, applications not meant to naturally run on cloud environments have to assess if they can rely on testing results over a virtualized system.

What we claim in the work presented is testing the cloud, formalizing the test methodology for cloud-based applications from the infrastructure point of view.

III. METHODOLOGY FOR DERIVING CLOUD INFRASTRUCTURE REQUIREMENTS

From the related work we found three things: firstly, the topic of cloud-based software testing is focused primarily on the STaaS model and not on the needs to the entire spectrum. Secondly, the topic usually surrounds the software testing capability as opposed to the needs to the infrastructure. Thirdly, there are no methodologies for determining which of the many options for cloud infrastructure usage are most effective for a given SuT. In this section we present a methodology for addressing the latter issue, which is focused on infrastructure concerns and is relevant for the entire spectrum of cloud-based software testing. The methodology is depicted in Figure 1.

The inputs software application properties describe the potential usage and resource consumption of the SuT, while the testing task model is a collection of all tasks in a specific testing process. As these vary on a case-by-case basis they are only described in the application study in Section V. The application testing effectiveness criteria and cloud test infrastructure patterns are however core to the methodology and are used as constant inputs in all case studies. These are described in Sections III.A and III.B respectively. Section III.C describes Activity 1 and 2 in the analysis section of the methodology, while Section III.D describes Activity 3 and the final outputs that consolidate the test infrastructure requirements.

A. Application Testing Effectiveness Criteria

The criteria are derived from an analysis of business goals related to software testing. Goal analysis in requirements engineering involves relating or transforming business goals (“what goals are to be achieved”) to functional and non-functional system components (the “how goals are to be achieved”) [15]. The overall business goal of any organization
is to make a profit (or to reduce deficits), which suggests that they have sub-goals of reducing capital and operational expenses (CAPEX and OPEX) and increasing their business activity. From a software development perspective cost reduction can be subdivided into reducing development costs and reducing operational costs, while increasing business requires quicker time to market and increased quality of software products and services. Tassey [4], in his NIST report on software testing (focusing on the transport industry), also suggests that these 4 criteria are affected when inadequate infrastructure is employed for software testing. These 4 criteria can be then further decomposed into 7 technical, infrastructure-oriented criteria (C1 – C7) for effective software testing, as depicted in Figure 2, and explained in the following.

These 7 criteria are with respect to infrastructure and test management:

C1. **Cost Effectiveness**: given that core business argument for cloud computing is reduction in total cost of ownership [16] the costs associated with testing must be minimal. Testing, bug-fixing and maintenance are considered to be sources of hidden costs in any software development process, such that the effort and resources required are often underestimated.

C2. **Simplicity**: errors of omission or commission in setting up and executing testing may cause bugs to be overlooked or introduced. Decreased complexity should be a benefit of choosing to test in a cloud environment. Assessing complexity includes the number of interactive steps and script complexity per task. Simplicity and cost-effectiveness are related and have an influence on the feasibility of the other requirements.

C3. **Target Representation**: running software in a cloud-based environment differs from running on internal, private resources without the technical capabilities of cloud-based technologies. These include virtualization, resource elasticity and web-based access, as well as resource sharing by users from different domains, on-demand provisioning of resources and dependency on “best-effort” Internet-based communication. If the test environment does not resemble these capabilities and characteristics the test can be deemed inconclusive.

C4. **Controllability**: conclusive testing requires control over the descriptive, structural and behavioral properties of the test infrastructure. Descriptive properties define the initial settings of the infrastructure, including the type of OS to be used, the amount of memory, storage and CPU, addresses and application-specific configuration parameters. Structural properties refer to the topology of the infrastructure and the software elements deployed, while behavioral properties refer to enabled functions, changing conditions and events that are triggered when certain conditions arise. A tester should be able to emulate and inject different types of capabilities, loads and faults to control the state of the infrastructure.

C5. **Observability**: the ability to control the properties of infrastructure is complimented by the ability to observe and verify those properties, leading to conclusive tests. For example, consider the case where increases in response times for an application service are detected even though the CPU capacity of the virtual resource appears sufficient. There is a need to control and monitor the utilization of the actual physical CPU that causes the response time delay to occur. This is an undesirable level of information to be disclosed for a cloud offering supporting production instances, but containable mechanisms are required for the purpose of testing.

C6. **Predictability**: the infrastructure and execution environment should behave as expected during testing, such that unexpected and unplanned test disruptions or anomalies do not arise. An unpredictable infrastructure leads to discarded tests, inconclusiveness and a need for additional validation or post processing of results. This adds to the complexity and uncertainty of the testing process, hence the overall cost. Note that the uncertainty associated with Internet connectivity is a factor when considering predictability as fundamental for testing in a cloud environment. However, it should be possible to simulate or emulate the uncertainty of the network connectivity with predictable impact on the test results.

C7. **Reproducibility**: the ability to repeatedly run tests under the same conditions, given that the same steps are followed, is also important for conclusive tests. Reproducibility is the ability to have minimal deviation in environment settings and procedures in comparison to the previous run, unless there has been significant correction or evolution in the system under test. Reproducibility and predictability are very closely related requirements in that sense. Reproducibility also serves to reduce complexity, as the data from previous test runs can be used as input to subsequent runs without compromising the results.

![Figure 2. The derivation of 7 infrastructure-oriented criteria (C1 – C7) for effective software application testing from business goals](image-url)
B. Cloud Test Infrastructure Patterns

The business goals and criteria from the previous section hold for any testing process executed on cloud-based infrastructure. However, there are various ways in which cloud-based infrastructure can be utilized for testing, each of which changes the feasibility of these requirements. These are captured as 5 different patterns shown in Figure 3.

<table>
<thead>
<tr>
<th>Pattern 1: Testing in the Cloud</th>
<th>Pattern 2: System under test in the cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern 3: Test-suite in the cloud</td>
<td>Pattern 4: Multi-site testing in the cloud</td>
</tr>
<tr>
<td>Pattern 5: Brokered multi-site testing in the cloud</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3. baseline + 5 deployment patterns for cloud-based software testing

Pattern 1 - Testing in the Cloud: this is a simple model of using a single cloud infrastructure provider for hosting the software under test and testing suite.

Pattern 2 - System under test in the cloud: in this case the tester places the system under test on a cloud provider’s infrastructure but hosts the test suite locally. The test payloads are sent over the Internet to the remote system under test. The software under test could be either owned by the tester or a SaaS solution under test.

Pattern 3 - Test-suite in the cloud: there exist providers of testing software in the cloud such that the test suite is hosted and controlled outside the domain of the SuT.

Pattern 4 - Multi-site testing in the cloud: more than one provider is used for distributing the SuT and the test suite in different domains.

Pattern 5 - Brokered multi-site testing in the Cloud: the same as Pattern 4 with the inclusion of an intermediary for brokering test and infrastructure management requests and operations. The broker [B] also provides support functions for resources selection, reservation and deployment.

C. Criteria and Complexity Analysis

Activity 1 of the process addresses the problem: given the set of patterns, effectiveness criteria and software application properties, which pattern will best satisfy the criteria for the particular application properties? This can be done using both qualitative and quantitative data, as each of the effectiveness criteria can be associated with concrete metrics. However we propose a qualitative rating scheme in Table 2 that can be used as an early analysis when all facts are not yet gathered.

| Activity 1 of the process addresses the problem: given the set of patterns, effectiveness criteria and software application properties, which pattern will best satisfy the criteria for the particular application properties? This can be done using both qualitative and quantitative data, as each of the effectiveness criteria can be associated with concrete metrics. However we propose a qualitative rating scheme in Table 2 that can be used as an early analysis when all facts are not yet gathered.

The 5 patterns from Section III.B are compared against the baseline per criteria, using the indicators in Table 2 to conclude Activity 1.

As seen in Figure 2., the criteria (C1 – C7) defined in Section III.A, contribute differently to the business goals, suggesting that there are differences in criteria weighting (cw). Furthermore, as shown in Table 3, there are different dependencies between the criteria, again changing their influence on the requirements analysis and on the individual criteria weights cw, where j is in the range 1 to 7. The criteria weight cw is calculated in Equation (1):

\[ cw_j = \frac{\text{num_dependencies}(j)}{\text{num_criteria}} + \frac{\text{num_contributions}(j)}{\text{num_business_goals}} \]

where \( j \) refers to the \( j \text{th} \) criterion in the set C1 – C7, \( \text{num_dependencies}(j) \) refers to the number of dependencies other criteria have on the \( j \text{th} \) criterion and \( \text{num_contributions}(j) \) refers to the contribution that the \( j \text{th} \) criterion makes to the business goals, considering the goal tree in Figure 2. In this case \( \text{num_criteria} = 7 \) and \( \text{num_business_goals} = 4 \).

From Table 3 criteria C2 - simplicity has the highest weight. This is reasonable as the simplicity of infrastructure management associated with testing contributes to cost reductions, quicker testing (hence time to market) and less likelihood of omission and commission errors during the process that can overlook or introduce bugs. Although C1 - cost effectiveness of infrastructure is a contributor to all business goals, the other criteria do not have a high dependency on cost. For this reason C6 - predictability has a higher weight than cost effectiveness, as it influences most other criteria. For example, with predictability it is possible to remove or automate steps in the process, as well as make assumptions about how to interpret data. The remaining criteria are critical for the effective execution of a testing process but do not have a significant contribution to the business goals. For example C5 - observability has the lowest weight as it is necessary for testing but does not directly impact on the business goals.
Activity 2 of the process is a complexity analysis of the testing process. This requires a detailed description of the tasks and their dependencies. Complexity of information systems operation processes considers the information, execution and flow complexity associated with the process. Similar concepts of process complexity are described by Keller et al. [17] and more recently by Xiao et al [18]. We take inspiration from these but aim for an approach that supports both early analysis and transition to quantitative analysis, once experimental data is obtained, as well as comparison of different patterns against the baseline. The complexity analysis methodology is captured in the following definitions.

Definition 1. A software testing process S is a collection of testing tasks s₁,…,sₙ that must be performed in a specific order to validate that a software system under test SuT satisfies its functional and quality requirements.

Definition 2. A testing task sᵢ ∈ S is any self-contained input action, decision, processing, output gathering or context shift that is performed within a software testing process.

Definition 3. The task complexity weight wᵢ expresses the information and execution complexity of the ᵐth task sᵢ in a process S, where wᵢ = wᵢᵢ + wᵢₑᵢ.  
- wᵢᵢ is the information complexity weighting and has three possible values in order of increasing complexity:  
  0 predefined information available  
  1 structured data and/or guidelines  
  2 unstructured data involved in task  
- wᵢₑᵢ is the execution complexity weighting and has three possible values in order of increasing complexity:  
  1 one time/ one command execution  
  2 automated execution  
  3 manual execution

The maximum value of wᵢ is hence 3 + 2 = 5, representing a manual task with unstructured data, while the simplest tasks have a weight of 1 with predefined information and performed once or with a single command.

Definition 4. The relative task complexity rating rᵢ is a comparison of the complexity of performing a task sᵢ in one pattern or setting against a baseline. The indicators in Table 2 are also applicable here. The subjectivity of these ratings reflects the priorities and capabilities of the analyst, such that the results of the analysis are tuned for the organizational perspective of the analyst. If the analyst is not the one that will perform the testing, they should carry out surveys with the actual testers to validate this subjectivity.

Definition 5. The relative process complexity rating Rₛ is the comparison of the complexity of performing a process S in one pattern or setting against a baseline. This is expressed in Equation (2)

\[ Rₛ = \frac{\sum_{i=1}^{n} wᵢ_rᵢ}{Rₛₘₐₓ * n} \]  

where n > 0 is the number of tasks in a process and Rₛₘₐₓ > 0 is a constant used to normalize the results such that 0 < Rₛ < 1. Using our ratings and indicators, Rₛₘₐₓ = 5 * 2 = 10. Rₛ can also be represented as a percentage (%) where the result is multiplied by 100.

D. Requirements Rationalisation and Consolidation

Activity 3 is then used to organize the final output of the process: the set of consolidated requirements and rationale for these requirements. The term “rationale” is used in software and systems engineering to capture “why” certain decisions have been made regarding requirements, selection, design or configuration of a system. Having an activity for rationalization causes typically implicit knowledge to be made explicit, such that the justification for decisions can be revisited at a later date [19]. Liang, Avgeriou and He [20] claim that rationale management is still under exploited in spite of its clear benefits. However, Dutoit and Paech [19] discuss that there is a significant, upfront investment of resources for rationale management, such that there is a need to justify this effort. For this reason our approach to rationalization of infrastructure requirements is incremental without an initial demand for rigorous analysis and hard quantitative data, which can be included as development and experience are advanced. The method of rationalization capture that best matches the analysis data we capture is the Questions, Options and Criteria (QOC) method from MacLean et al. [21]. In Activity 3 of our framework there are 2 questions asked, with the following options and criteria:

- Q1: which cloud usage pattern is most effective for the testing process?  
  - Options: the baseline and 5 patterns P₁ – P₅  
  - Criteria: the 7 effectiveness criteria plus the relative process complexity rating Rₛ per pattern

- Q2: what improvements in support for testing tasks should be investigated?  
  - Options: execution and information simplification including removal, decomposition and automation of tasks, as well as information input reduction.  
  - Criteria: reduction in execution and information complexity weighting wᵢ per task (Definition 3).

The benefits of this rationalization are the ability to identify and justify the most appropriate pattern for supporting the testing process but also the ability to identify where further automation and information simplification investigation might be required.

IV. APPLICATION STUDY

We have selected an application study in the area of PaaS as there are no established models or best practices for software testing in this domain. Furthermore the multiple layers and dependencies in such applications provide a good background for evaluating the relevance of the taxonomy and associated analysis methods. In order to ensure that the analysis is justifiable and objective, we implemented a basic SuT and
worked with HP Research’s Cloud Service [22] to practically validate the ratings used in each analysis.

A. Activity 1: Criteria Analysis

The methodology starts with gathering information about the application’s properties and its target environment. Figure 4 illustrates the case study showing 3 PaaS providers serving 2 different client organizations. Each provider enables concurrent access to their autonomous platforms by registered client organizations, their application developers and application users. In Figure 4, client organization 1 has a larger application with more users and requires more resources and scalability. Besides functional testing, performance/load testing is critical as there are service level agreements between the various providers and client organizations. The application components are distributed across three containers in provider 1 and 2, with provider 3 acting as a redundant resource provider.

Figure 4. Illustration of deployment for case study

As a container we use an Open Services Gateway interface (OSGi) [23] platform such that application components are implemented and deployed as Java OSGi bundles. We then assume that all providers adhere to OSGi and that these bundles can be migrated seamlessly across providers. We also associate each container instance with a single VM instance. We assume and do not restrict that bundles from different organizations can run in the same container instance. For example, in Figure 4, Provider 1 has a Container with bundles from Organization 1 and Organization 2. Nevertheless, the security and performance isolation requirements for the cloud user experience should still hold.

Activity 1, where the infrastructure criteria analysis is performed. The results are presented in Table 4 and discussed subsequently.

As all patterns involve outsourcing to 1 or more cloud providers with dedicated infrastructure, all patterns satisfy the cost-effectiveness criteria. P5, the brokered multi-site testing pattern, turns out to be the only one that shows some improvement over the baseline for this particular type of application. This is firstly because of the simplicity that a central point of control, such as a broker, gives to multi-site testing. Secondly, the multi-site cloud is the best representation of the target production infrastructure within which this application would be deployed. However, it is noted that there would be significant losses in observability, as the broker masks what is happening at the infrastructure. The worst option from the perspective of criteria analysis would be P1, complete testing in the cloud approach, as this is not representative of the target environment and loses simplicity, as the infrastructure, resource management practices and scripts must be adapted to the particular cloud provider.

B. Activity 2: Testing Process Complexity Analysis

The testing process for this particular application setting has 22 steps including specification of infrastructure requirements, installation of the test environment, compilation of the Java OSGi bundles and their deployment on the infrastructure.

Table 4. Results from Activity 1 criteria analysis (see Section II.C)

This description of the application, along with the set of 5 patterns (Figure 3) and 7 criteria (Figure 2) serves as input

Figure 5. Testing tasks and results from complexity analysis

Figure 5. shows the 22 steps (S1 – S22), the weight w_i per task, a plot of the weighted relative complexity rating (w_i * r_i) per task in Figure 5. (A) and a plot of relative process complexity rating per pattern P1 – P5 in Figure 5. (B). In this analysis P3 appears to be the best option. This is because heavily-weighted tasks such as configuring the load testing software are simplified, as testing service providers specialize in exposing interfaces and utilities for managing tests. P5 appeared to be the best choice in the infrastructure effectiveness criteria analysis but only places 3rd in this case. This does not invalidate the first analysis but suggests
improvements to P5’s testing support and improvements in P3’s infrastructure management, as inputs to Activity 3.

C. Activity 3: Rationalization and Consolidation

The results from Activity 1 and Activity 2 provide some concrete suggestions for addressing Q1 and Q2 in the methodology. Firstly, it was found that P5 is the most effective for the testing process, using the 7 weighted criteria. However, the fact that P3 emerged better than P5 for reduction in test process complexity suggests that there are cases where support can be improved. This also validates the investment in software testing services in the cloud emerging as offering. Nevertheless, there was only a 5.45% test support improvement by P3, suggesting that there are a number of tasks that could benefit from better infrastructure support.

V. CONCLUSIONS

The methodology described and demonstrated in this paper features a taxonomy of 5 patterns and 7 criteria for addressing this decision problem. We have also treated the topic of software testing and cloud computing in a more holistic and infrastructure-oriented manner. This goes beyond the state of the art, where the focus has been on the basic model of providing software testing services. It is however noted that the software testing services in the cloud model, P3 in our set of patterns, provided the best improvement over the baseline in the test process complexity analysis. This can be seen as a justification of this offering. We believe that there is value in pursuing the broker model of test infrastructure provisioning, given that it showed the best (only) improvement over the baseline across the 7 weighted criteria. There may be other criteria included in the analysis such as correctness, as the methodology is defined with extensibility in mind.

Future work is towards simplifying tasks in a testing process with capabilities of the infrastructure. We propose that the brokered pattern (P5) with explicit support for test management simplicity is ideal for testing cloud-based applications. The BonFIRE project [24] provides an experimental infrastructure that enables practical investigation of this hypothesis. It would also be of value to extend and refine the patterns with generic components such as load and fault generators that can be distributed in different ways, considering different types of testing including fault, stress and load testing.

ACKNOWLEDGEMENTS:

We thank our colleagues from HP Research for their invaluable input to this paper. In particular Frederic Gittler for his discussions from which the criteria evolved and Luis Vaquero for support in accessing and using the HP Research Infrastructure service. This work is supported by the EU-FP7 Project BonFIRE 257386.

REFERENCES