Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 5 (2011) 513–520

Procedia Computer Science

The 2nd International Conference on Ambient Systems, Networks and Technologies (ANT)

Cloud Services Testing: An Understanding

Atif Farid Mohammad[a], Hamid Mcheick[b]

[a] Department of Computer Science, University of North Dakota, Grand Forks, ND USA
[b] Department of Computer Science, University of Quebec at Chicoutimi, Quebec, Canada

Abstract

There is a mammoth quantity of Cloud Services being deployed on the Cloud, nowadays. Everyday users and customers use these services to fulfil their needs. The use of Cloud Services is getting more and more complex with the pace of time. It is a possibility that several recurrent service requests cannot be fulfilled using just one Cloud service. Cloud services are usually composed manually, which is a time consuming and monotonous task. We can find several numbers of successful methods for automatic Cloud service composition, the main issue with that is the lack of test environment with some standards to compare and evaluate these methods. This research work is about a short survey to explore Cloud Services testing methods. This study compares several software testing researches and pose questions for further research work to find Cloud suited testing techniques for the software testers. This survey paper poses few questions to the Cloud computing research community to concentrate and find suited answers for software testing community.

Selection and/or peer-review under responsibility of Prof. Elhadi Shakshuki and Prof. Muhammad Younas.

Keywords: Architecture, Cloud Computing, Cloud Services, Services, Service Oriented Architecture, Software as a Service.

1. Introduction

Cloud Computing is a combination of several services, such as Infrastructure-as-a-Service (IaaS) like Amazon™ Web Services provides virtual servers with unique IP addresses and blocks of storage on demand [1]. Whereas Platform-as-a-Service (PaaS) is a set of software and development tools hosted on the provider's servers, such as Amazon Elastic Cloud [1], EMC Atmos [2], Aptana [3] and GoGrid [4] are providing these services preventing users the mammoth costs of buying hardware, software and related technology as well as, to maintain and supporting their IT infrastructures. The third major participant in

Selection and/or peer-review under responsibility of Prof. Elhadi Shakshuki and Prof. Muhammad Younas. doi:10.1016/j.procs.2011.07.066

Cloud Computing is Software-as-a-Service (SaaS) is a model whereby software is provided by a vendor, such as when an ISP or a cable company provides an Internet connection and charges its services for a certain period, in a monthly or quarterly fee for the provided service. The service provider in this case is the IT infrastructures of a hosting company, which runs the software at its data centre and lets the consumer, uses its services under a certain agreement among service provider and user. SaaS is an on-demand commodity and is available in Cloud from several service providers. SAP™ and Oracle™ are good examples of SaaS providers. The users pay only for the services they use.

Cloud Services Technology [5, 6] can be considered as the key technologies to create Cloud Service(s). A Cloud service can also be defined as a set of functionalities that can be invoked by a user or a Cloud application request to perform some required task(s) through the web [7]. Cloud computing is hypothesized to be a paradigm shift or an evolution in the IT industry. Quality computing services can be provided to an everyday user by consolidating many datacenters and software as a service providers to reduce computing costs. There are three foremost units, and those are Cloud Applications, cloud Storage Providers, and end users [8]. Briefly speaking, several methods have been developed for automatic cloud service composition. But the main issue with that is the lack of test environment. This paper survey Cloud Services testing methods and compares several software testing researches and pose questions for further research work to find Cloud suited testing techniques for the software testers. This article is arranged as follows. Section 2 describes a background of cloud computing applications and tools. Section 3 gives an overview of traditional software testing. Software engineering and testing are described in section 4. Sections 5 and 6 explain SOA and testing as well as Cloud service testing respectively and pose few questions to the cloud computing community. We conclude in section 7.

2. Background

The regeneration of replicas for the replacement of crashed services was first proposed by Pu [9]. In the context of a cluster communication systems [10, 11] provide details on automatic reconfiguration and regeneration of service replicas, where as author of [12] put the focus on regeneration in a peer-to-peer wide-area storage system. Another comparable mechanism for migration, placement, and monitoring of components in the cloud are provided as a proposal [13] to initiate more research work. Such Cloud applications provide the essential methods that are required to support service deployment in Cloud, which can be public, private or hybrid. However, this research work has a focus on regeneration of new services to recover the availability and reliability of a failure. The study done by Yu and Gibbons [14] provides theoretical notation that replica replacements can extensively impact Cloud application availability.

Generic Cloud services resource discovery system is discussed by Albrecht et al. [15], which describe enabling querying for available resources by taking a database-like approach; they use an algorithm to find mappings for centralized group-finding using optimization techniques. Many other frameworks [16, 17] have provided on finding optimal placements for virtual machines under a variety of constraints. By maximizing [18] the utility of services via deployment decision making has been investigated in an algorithm, which has been formulated by the authors that is based on calculating weighted sums as the usefulness of the alternative configurations. However, the consequential advancement in their research work is not effective on computational background and can be considered as an assessment to illustrate that deployment decision making. Conversely, these approaches rely on a centralized optimization that needs to be processed through the entire state-space of decision options.

HP Labs initiative of the SmartFrog [19] deployment and management framework illustrates Web or Cloud Services as a collection of components and applies a distributed engine comprised of daemons running on every node in a network, which can also be called Cloud. These services can be activated and managed together with their configuration parameters to deliver the desired services even in Cloud

Applications. The authors of [20], have presented resource allocation algorithms to solve service distribution problems on the basis of biologically-inspired methodologies.

3. Traditional Software Testing

It is not clear how the software testing techniques, suggestions, methods, and tools should be applied to the practice of testing on Cloud systems, because we do not have an explicit and generally accepted understanding of Cloud testing of what can be included in the practice of Cloud system testing. Traditional software testing methodologies are mostly based on best practices of testers and standards than on theory. It is mentioned by Kitchenham et al. [21] in his research work that there is no well-defined theory about how software engineers find and/or introduce defects into software systems, nor any theory is provided about how the testers are to recognize those flaws or bugs.

If testers of an XYZ Software Testing Corporation write a proposal after testing one enterprise system improvement proposal, may not be appropriate in another business venture due to the variation of the process content among several corporate sectors. There can be several factors analysis can have an effect on testing procedures, which can be complicated. Taipale et al. [22] divided these factors into two rational premises: factors that have affects on testing and related procedures and second rational of these premises, which can affect testing know-how and another business venture. This paper contemplates on the first premise, factors affecting testing and related processes. In the available research work on software testing methods diverse factors affecting testing processes are underlined. These factors include:

i) Testing and test contributions in the design and development processes, ii) Testing of management complexity, iii) Testing on the basis of risk, iv) The interaction between design, development and testing, v) Testing of software units, and vi) Testing adjustment as per the business policies and procedures.

Software testing should be incorporate in the design and development processes, so that testers can design and develop tests for requirements provided by the end-users and that developer's analysis [23]. The testing is a parallel activity to the development process and can be complicated issue. It is mentioned by Baskerville et al. [24] that a parallel test is conducted by the software developers to run their testing or quality assurance with activities, such as design and development process are on as mutually adjusted processes.

Strategic management of complexity has been discussed by Salminen et al. [25]. Most of the testing processes are related to testing strategies, where these strategies are defined the contents of testing. For example, the risk-based testing strategy, which puts focus on spending more time on critical functions [26]. Co-operation and coordination of schedules [27] requires communication and interaction between development and testing processes. The conflict between the testers and project managers is mentioned in the survey by Cohen & al. [28], which was the time allocation between design, development and testing phases. It is clear from all of these references, that many factors affect testing processes. Cloud services testing is a new field with several unexplored challenges, this paper is sort of an initiative to find a comparison between Cloud computing testing and earlier paradigms, however, it is difficult to reach a unanimously agreed definition for cloud services testing criterion.

4. Software Engineering and Testing

When man first started to construct dwellings, the task was a somewhat haphazard process. Small buildings were created easily, and problems that cropped up later could be fixed quickly. The techniques used to create small buildings were abandoned as shelter requirements and testing of the prototype of early buildings became more complex, when the phenomena of prototype came into being due to the old methods just wouldn't work anymore, as larger structures began to collapse upon themselves, much to the disappointment of the occupants. Out of these disasters, architectural engineering was born. The creation of modular construction components as a result of standardized measurements is at least as old as organized civilization.

Early construction tools were limited to stone and mud, but, even with these limitations, fantastic creations were possible. One need only look at the great pyramids [29, 30] of Khafra and Cyops at Giza to see the accomplishments of ancient civilizations. Craftsmanship, time, and patience were the key. Over time, as materials began to be prefabricated, more flexible and less labor-intensive techniques were introduced. Now, the modern skyscraper, with its prefabricated steel girders and pre-measured components, serves as a lesson for what can be achieved with standardized materials and improved engineering techniques. The Sears Tower took far fewer people and less time to construct than the pyramids. Admittedly, it won't last as long, but then again, it was never meant to, same scenario applies to the Cloud Services. Due to the rapid change in user and business needs, new services are being composed, tested and being deployed almost every day basis in the Cloud.

What does all of this have to do with Cloud applications? Everything! The history of software development is analogous to that of construction. Early software projects were small, requiring only a few developers and testers on the site of development and deployment, customized code, and little time. These programs served their users well. However, as system requirements increased, the old development methods used were no longer sufficient.

The art of software development began with the first computer [31], the ENIAC, just as knowledge about building began with man's first structures. However, the theory of software development started as an idea before the first machines were created. The EDVAC Report written by Von Neumann in 1994 outlined the first interface between hardware and software. Most modern hardware and its underlying software have been based on Von Neuman's basic design [31]. Like the stones of the pyramids, once machine code became as well, ALGOL [32] was among the early standouts in the area of language development. We will see that these early languages, while useful and powerful in their time, now appear as the split trees of log cabin construction when compared to the modular and object-oriented languages of today.

In any current Cloud service testing, end users' participation is even more active and direct. These end users can be either individuals or corporate users and have become a powerful and indispensable part of the Cloud testing team for Cloud applications and for the Cloud services providers. Consider the typical software PC development project of the 1980s. It was probably very small and employed by, at best, a handful of users. It manipulated development-level data and produced a small group of reports, which used to be tested in-house by the programmers themselves. Due to the limitations of the hardware and software, most large applications resided on a mainframe or minicomputer. Any interoffice connectivity or data-handling and application testing capabilities rested with programmers and in-house end-users in the MIS Department who were required to use tools and write applications that had life spans of up to 15 years. COBOL [33] emerged as the language of choice for most business software development projects.

Once a programmer wrote an application, he was expected to test and maintain it. Since most traditional software applications reflected the individual programmer's own abstraction, it was difficult for others to maintain these applications. New maintenance programmers, hired as replacements for the original developer, invariably encountered difficulties, because the application didn't measure up to the new developer's abstraction of how the application should work. If you were to magnify this problem as it applies to large-scale application development, you would see why these significant issues must be addressed. Programming as well as system testing on a large scale presents issues not normally addressed in small development projects. As you add programmers to a project, you begin to realize just how many problems exist in traditional development methodologies.

5. SOA 3.0 and Testing

The Ten Commandments [34] of SOA are given below to ponder on, before we continue to explore SOA 3.0 or ASOA methodology.

1. *Thou shalt not disrupt the legacy system.*

2. *Thou shalt avoid massive overhauls. Honor incremental partial solutions instead.*
3. *Thou shalt worship configuration over customization.*
4. *Thou shalt not re-invent the wheel.*
5. *Thou shalt not fix what is not broken.*
6. *Thou shalt intercept or adapt rather than re-write.*
7. *Thou shalt build federations before attempting any integration.*
8. *Thou shalt prefer simple recovery over complex prevention.*
9. *Thou shalt avoid gratuitously complex standards.*
10. *Thou shalt create architecture of participation. The social aspects of successful SOA tend to dominate the technical aspects.*

SOA 3.0, which is also known as ASOA or Achievable Service Oriented Architecture is a novel approach that provides an efficient and cost effective methodology for organizations to achieve their required services fast and easy. This section puts light on the adaptable approaches, which can be adapted for software design of required services front-end applications. It is very important to know the application design methodologies, such as abstraction, modular application design and information hiding used to understand the available software structure. This understanding can be utilized as the bases to develop wrappers for legacy systems to get data converted in to XML for the use of latest develop services.

The abstraction used in creating software designed with structured programming in mind did not focus its consideration on maintainability or application reuse-testing as a service for long term basis. The focus was instead placed on the application at hand to resolve an issue for a certain timeframe. With users and management teams breathing down the software professional's neck, it was easy to lose focus on the long-term needs of the developers, testers, users, and management alike. If the designer management instead put some focus on the long-term view, they were more likely being able to achieve professional goals and meet the user's needs of the future as well.

The understanding of legacy approaches is the key to be successful to provide an Achievable SOA or SOA 3.0 solution. This lacking of structured programming approach, the abstraction used to create the initial templates for software developments are flawed if viewed in the light of reusability, once tested by the in-house developers working closely with application developers. These abstractions lead to later modification of the template for these in use applications of the legacy systems used in the banking industry in specific and in other industries in general. While at first these modifications found by the system testers seem minor, one must realize that each change to a template creates a new version of the code. This versioning leads to a larger and larger volume of source code to maintain which, in turn, increases long-term software costs.

SOA 3.0 or ASOA as mentioned by the author [34] provides the saving of this cost. If we again use the analogy of programming to construction, the pyramids of Egypt typify the limits of structured programming. Each pyramid created taught the Egyptians something about construction. This being the case, each pyramid was slightly different. Even those built around the same time period have their slopes set to different angles. What is the likelihood that stones cut for one pyramid could be used in another? Slim indeed. Template-based design, development and testing approach to design these legacy systems also proved to be inflexible. As templates gradually improved, legacy systems based on older versions were not easily updated by the developers, due to the lack of proper documentation available. It is a reality even in 2011 that software engineers are extremely unlikely have the time to revisit older legacy systems applications in order to add improvements designed and get developed for newer systems. SOA brings a light to resolve these issues. The SOA 3.0 approach presented in [34] makes both the management and IS/IT departments to work together to achieve a workable, efficient and cost saving solution for the organization using available resources. The prevailing attitude says, "If it aren't broke, don't fix it." This is wise given the constraints of the methodology. The Egyptians never went back to improve older pyramid designs, they just built them bigger and better the next time. This is exactly the same approach, which SOA, since conceived and is in practice of utilization.

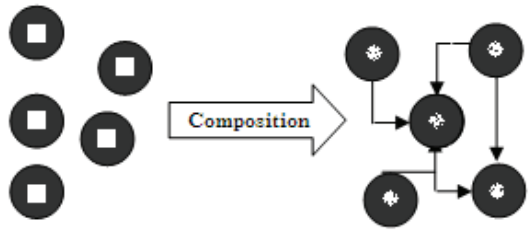

Figure 5.1: Composability of the available legacy components into Cloud Services

SOA provides composability of the legacy components to be aligned and used with minimal new developments. This composability is given in Figure 5.1. Abstractions can be reused to create a new service by decomposing the legacy modules needed to complete a SOA 3.0 project. Typically, these legacy components are not composable, that is they cannot be easily reused in dissimilar combinations to accomplish a different objective and need an extensive testing by both in-house and some external software testing teams.

Since service components designed and developed to a specific abstraction was not originally designed for reuse, composability was in general not a goal of the management and software engineers at the time of designing the legacy systems. This limitation of structured programming methodology of software engineering provided us the basis for another paradigm in larger and complex software projects; this is known as modular methodology or approach to design new systems.

Software engineers realized the need of creating truly reusable software components. This realization was the basis of modular software construction while incorporating agile testing of these components incorporated before deployment. Let us revisit this approach by realizing that software design and development is as much a tools building process, as it is a Cloud service development process, software engineers can significantly decrease the time spent on new development and post implementation maintenance, which now involves testing as an essential part to make sure that the end users do not find errors, while processing or requesting their needs using the service. Software engineers are some of the last few practitioners that create their own tools. Independent software engineering and SOA consultants should also realize that this includes them as well. Each service required currently by any organization should be thought of as an opportunity to learn new skills and create new designs of using legacy systems as methodology tools that can be reused in later service design and development for any other department of other industrial concern.

6. Cloud Service Testing

Cloud computing is the availability of logical computational resources, such as data, software (SaaS) and network accessible to a user on demand on anywhere basis rather than from a local computer. Users or clients can perform a task using a service, such as preparing an inventory report or performing word processing, provided through such cloud based computational resources within a browser and with service that performs the required functions and/or outputs. Cloud service testing for any of its' feature such as security involves identifying a set of predefined policies, to design the test cases to cover maximum user anticipated needs. There is an incomplete list of such questions is given below:

A. What can be the service security policies?
B. How these security policies to be modelled to get results on the base of specification based testing?
C. How the testing teams exhaustively test a service against its security policies?
D. What is the exhaustive usability coverage criterion?
E. Will adequate coverage be more suited than extensive service usability coverage"?
F. What is the adequacy criterion?

G. What is the measurement criterion to evaluate the quality of such adequacy?
H. What are the adequate test inputs to be generated to achieve this adequate coverage?
I. What can be the test oracles generated to judge whether a test outcome is correct or not?
J. What are the challenges in formulating test oracles for service security?
K. On the basis of test outcomes, how can the development team identify the problem sources?
L. How does the instrumentation for Cloud service security differ from traditional networked systems?

It is vital to provide a protective shield to the cloud service developers and providers, by preventing attentive observers for the testing procedure from acquiring internal or premature information about a Cloud service composition by participating in the testing process as an external user. In particular, we should find answers to the following research questions.

1. How a "user" is defined in a testing procedure?
2. What are user characteristics, and how can we map these characteristics to the testing process?
3. What information about the service component(s) should be given to the internal test users, and what should be hidden from them?
4. What information about the service component(s) should be given to the external test users, and what should be hidden from them?
5. By making sure the protection of hidden information, how to provide sufficient information to facilitate testing to the testing team?
6. What can be the classification of internal and external test users in certain groups, with such grouping, what can be the trust level?
7. What can be the scale of measurement to monitor users' groups' trustworthiness?
8. What can be the criterion to modularize the user-related service(s) to assist competent testing, which is safe and ensures hidden information confidentiality and end-user satisfaction?
9. What program (service) code slices to be given to the testing teams?

The Cloud service design and development as well as testing of available services have shown considerable diversity from the traditional in-house system development for an internet enabled monotonous application. The list of questions given is not complete and extensive, exactly as the testing of a software can never go extensive, to save both time and cost to the end-user.

7. Conclusion

As described in the paper, though there are tremendous advantages in using cloud-based systems, there are yet many realistic problems of Cloud service(s) testing need to be solved. Cloud is a union of several hardware and software platforms. There are several hybrid technologies being utilized to provide the services in the cloud by many service providers. This article sheds light on new insights into software testing in the new era of Cloud system development and testing. It is not intended to solicit consensus, rather this analysis can help rendering technical challenges and research problems in Cloud computing as a central element. It is easy to trace back many research problems in earlier works in related areas of software testing; there can only be found some available techniques, which can provide comprehensive and competent solutions suitable for the area of Cloud Services testing. The significance of Cloud is now a well-known fact and it is for sure that the researchers need to devise novel designs and techniques for cloud computing with the pace of time to assist end-users requirements.

References

[1] http://aws.amazon.com; Accessed on February 21st, 2011
[2] http://www.emc.com/products/family/atmos.htm; Accessed on February 21st, 2011
[3] http://docs.aptana.com/docs/index.php/Aptana_Cloud Docs_Wiki; Accessed on February 21st, 2011

[4] http://www.gogrid.com; Accessed on February 21st, 2011
[5] Berners-Lee T. Services and semantics: web architecture; 2001, Available at http://www.w3.org/2001/04/30-tbl
[6] Bussler C, Fensel D, Maedche A. A conceptual architecture for semantic web enabled web services. SIGMOD Rec 2002;3(4):24–9
[7] Tsur S, Abiteboul S, Agrawal R, Dayal U, Klein J, Weikum G. Are web services the next revolution in e-commerce? In: Proceedings of the VLDB conference, Rome, 2001, p. 614–7
[8] Michael A. Armando. F, Rean G. Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. UC Berkeley Technical Report UCB/EECS-2009-28, February 2009
[9] C. Pu, J. D. Noe, and Proudfoot. A.. Regeneration of replicated objects: A technique and its eden implementation. IEEE Trans.actions on Software Engineering, 14(7):936–945, July 1989
[10] Meling. H. and J. L. Gilje. A Distributed Approach to Autonomous Fault Treatment in Spread. In 7th European Dependable Computing Conference, Kaunas, Lithuania. IEEE CS. May 2008
[11] Meling. H. Montresor, A. B. E. Helvik, and O. Babaoglu. Jgroup/ARM: a distributed object group platform with autonomous replication management. Software: Practice and Experience, 38(9):885–923, July 2008
[12] Yu. H. and Vahdat. A. Consistent and automatic replica regeneration. ACM Trans. on Storage, 1(1):3–37, Dec. 2004
[13] Elmroth. E and Larsson. L. Interfaces for placement, migration, and monitoring of virtual machines in federated clouds. In 8th Int'l Conf. on Grid and Cooperative Computing (GCC 2009). IEEE Computer Society Press, Aug 2009
[14] Yu. P. and Gibbons. P. B. Optimal inter-object correlation when replicating for availability. Distributed Computing, 21(5):367–384, Feb. 2009
[15] Albrecht J. Oppenheimer D. Vahdat A, and D. A. Patterson. Design and implementation trade-offs for wide-area resource discovery. ACM Trans. on Internet Technology, 8(4), Sept. 2008
[16] Verma A. Ahuja P. and Neogi A. pmapper: power and migration cost aware application placement in virtualized systems. In 9th Int'l Conf. on Middleware, pages 243–264, Dec. 2008
[17] Joshi K. Hiltunen M., and Jung G.. Performance aware regeneration in virtualized multitier applications. In Workshop on Proactive Failure Avoidance Recovery and Maintenance, Lisbon, Portugal, June 2009
[18] Kusber R., Haseloff S., and David K. An approach to autonomic deployment decision making. In 3rd Int'l Workshop on Self-Organizing Systems, Vienna, Austria, Dec. 2008
[19] Sabharwal R. Grid infrastructure deployment using smartfrog technology. In Int'l Conf. on Networking and Services, Santa Clara, USA, pages 73–79, Jul 2006
[20] Heimfarth T. and Janacik P. Ant based heuristic for os service distribution on adhoc networks. Biologically Inspired Cooperative Computing, 2006
[21] Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., Emam, K. E., and Rosenberg, J. Preliminary Guidelines for Empirical Research in Software Engineering. IEEE Transactions on Software Engineering, 28, No. 8 2002, 721-733
[22] Taipale, O., Smolander, K., and Kälviäinen, H. Cost Reduction and Quality Improvement in Software Testing. In Software Quality Management Conference, Southampton, UK, 2006
[23] Graham, D. Requirements and testing: Seven missing-link myths. IEEE Software, 19, 5, 2002, 15-17
[24] Baskerville, R., Levine, L., Pries-Heje, J., Ramesh, B., and Slaughter, S. How Internet Software Companies Negotiate Quality. Computer, 34, 5 ,2001, 51-57
[25] Salminen, V., Yassine, A., and Riitahuhta, A. Strategic Management of Complexity in Distributed Product Development. In NordDesign 2000, Copenhagen, 2000
[26] Amland, S. Risk Based Testing and Metrics: Risk analysis fundamentals and metrics for software testing including a financial application case study. The Journal of Systems and Software, 53, 2000, 287-295
[27] Taipale, O., Smolander, K., and Kälviäinen, H. Factors Affecting Software Testing Time Schedule. In the Australian Software Engineering Conference, Sydney, 2006
[28] Cohen, C. F., Birkin, S. J., Garfield, M. J., and Webb, H. W. Managing Conflict in Software Testing. Communications of the ACM, 47, 1 2004
[29] Lepre, J.P. The Egyptian Pyramids: A Comprehensive, Illustrated Reference. McFarland & Company. ISBN 0899504612. 1990
[30] Levy, J. The Great Pyramid of Giza: Measuring Length, Area, Volume, and Angles. Rosen Publishing Group. ISBN 1404260595. 2005
[31] Goldstine, Herman H. The Computer: from Pascal to von Neumann. Princeton, New Jersey: Princeton University Press. ISBN 0-691-02367-0. 1972
[32] P. Wegner. Programming Languages The First 25 Years. in IEEE Transactions on Computers, Volume 25, Issue 12, pp. 1207-1225. December 1976
[33] Wegner P. Programming Languages The First 25 Years. in IEEE Transactions on Computers, Volume 25, Issue 12, pp. 1207-1225. December 1976
[34] Atif Farid Mohammad; SOA and Use of System Requirements Design - SRD: An Era of SOA 3.0. International Conference on Software Engineering Research and Practice. Las Vegas, NV July 2009