Contents lists available at Science-Gate

International Journal of Advanced and Applied Sciences

Journal homepage: http://www.science-gate.com/IJAAS.html

# A scenario-based distributed testing model for software applications

Mirza Aamir Mehmood [1, 3], Azhar Mahmood [1,*], Muhammad Naeem Ahmed Khan [1], Shaheen Khatoon [2]

[1]*Department of Computer Science, Shaheed Zulfikar Ali Bhutto Institute of Science and Technology, SZABIST, Islamabad, Pakistan*
[2]*College of Computer Science, King Faisal University, Al Hassa, Saudi Arabia*
[3]*Balochistan University of Information Technology, Engineering and Management Sciences, Quetta, Pakistan*

ARTICLE INFO

*Article history:*
Received 10 June 2016
Received in revised form
16 October 2016
Accepted 19 October 2016

*Keywords:*
Distributed testing
Software testing
VandV
Distributed application testing

ABSTRACT

Modern software applications are getting more complex in order to provide better service and quality. This complexity has given birth too many challenges for software testing such as functional discontinuation and detection of system level defects. Existing testing techniques which are based on test cases are time consuming and unable to offer higher confidence on quality of products for these complex applications. Distributed testing frameworks could be used to test complex software but these frameworks do not provide a global picture of testing activities and application status. This poor visibility results in poor control over testing activities. In this study we have proposed a distributed testing model (DisTest) using scenario-based testing technique. DisTest could be used with any COTS test automation tool and could be employed at any testing level. Result shows DisTest provides better visibility and control on testing activities and view of application status.



## 1. Introduction

Software testing is a defect detection technique and integrated phase of Software Development Life Cycle (SDLC). It is employed with other defect prevention techniques such as static analysis. Fig. 1 shows different testing levels and strategies based on their characteristics.

In order to select type of testing project managers need to consider cost and quality to deliver high quality software within budget. Testing is a costly task and it is estimated that mature organization spent 20-50% efforts on software testing (Desikan, 2006). To cope with these challenges there is a growing body of research based upon nature of application among them.

Distributed software testing is much more difficult than testing a conventional desktop application since distributed testing environment present many challenges for software testing e.g. dynamically evolving system architecture, stochastic behavior, complex component interactions, functional discontinuations, failure under load and stress condition (Mogul, 2006), effects of correlated failures (Haeberlen et al., 2005), bottlenecks arising from complex network topologies, availability, scalability etc. All of these issues make distributed software testing very challenging and difficult to accomplish (Gupta et al., 2011).

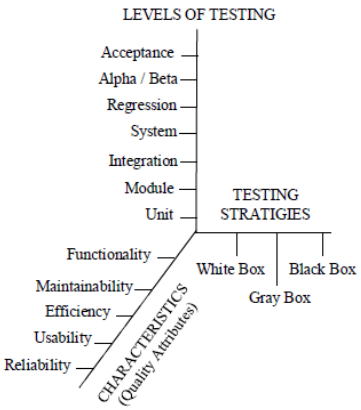


**Fig. 1:** Quality attributes, levels and strategies of software testing (Tómasson, 2011)

Existing distributed testing frameworks cannot deal with testing challenges outlined above. In this study a model is proposed for automated distributed software testing. DisTest is intended for test engineers thus black box methodology is employed.

* Corresponding Author.
Email Address: azharmahmood8@hotmail.com (A. Mahmood)
https://doi.org/10.21833/ijaas.2016.10.011
2313-626X/© 2016 The Authors. Published by IASE.

It can handle testing challenges such as stochastic behavior and functional discontinuation. This model can be used with any commercial off-the-shelf (COTS) test automation tool and could be deployed in any environment where Java is supported. Proposed model can be applied at any level of software testing, such as integration testing, where not all the components are fully developed and at the system level, where it can uncover system level defects which existing techniques may fail to reveal.

Proposed distributed testing framework is developed by using concept of scenario based testing. It is a software testing technique employed when black box testing strategies are used. A typical scenario is shown in Fig. 2 where user is login to system and performing different task by considering system as black box.

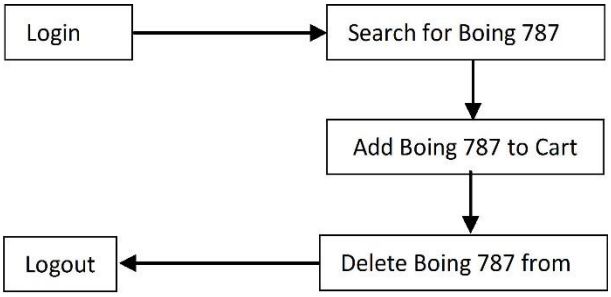


**Fig. 2:** A test scenario

Scenario-based testing technique has many advantages over other techniques. A scenario is a hypothetical story, used to help a person think through a complex problem or system. A scenario test is based on a motivating story about how the program is used, including information about the motivations of the people involved. Stakeholders believe that scenario not only could happen in the real world; stakeholders also believe that something like it probably will happen (Kaner, 2003). Scenarios are based on stakeholders prospective and their priorities of functionalities. Test scenarios are used to establish or enhance level of confidence on quality of software to verify complete operational flow of a system. Scenarios give better test coverage by testing all part of functionalities of the system under test with a measureable objective and goal. This way we can uncover defects in the existing design which left uncovered using any other techniques. There could be infinite number of test scenarios but scenario prioritization and selection is easy as it is based on user's priorities.

## 2. Literature review

Software testing itself bears a lot of challenges and distributed testing adds another layer of complexity in the testing process. This complexity also effects construction of test oracle and makes it difficult. Hierons (2012) has identified two important challenges of distributed testing have been identified i.e. controllability and observability. In distributed testing, test cases are executed on remote nodes and outcome of these test cases are local to the remote node. Testing in this fashion does not provide global picture, making observability difficult and also results in poor control over testing activities. With the help of mathematical model it is established that Oracle problem is NP-hard for the strong conformance while for weak conformance Oracle problem can be solved in polynomial time.

Dhavachelvan et al. (2006) observed that existing testing techniques in which test cases are developed against use cases is time consuming and inefficient when testing large and complex applications thus suggested using agent bases testing framework. Study aimed to address two important challenges of testing i.e. time reduction and effectiveness improvement. After experimentation it is observed that by using this framework time and cost is reduced. Paydar and Kahani (2011) has also suggested using agent based system for testing web applications. Another solution for reducing cost and time is using test automation tools. Developing automated test scripts requires test engineers to learn scripting language which is time consuming. To address this issue Gupta and Bajpai (2014) proposed keyword driven testing in which a key value pair based dictionary is developed for writing test cases. Thus learning cure is reduced and test case development time gets reduced.

Alvaro et al. (2012) employed white box testing strategy. Test cases are developed in Bloom language therefore developers don't have to learn another language. It has a built in test execution mechanism and is capable to generate test date. The main drawback of this framework is that it can test only those applications which are developed using Bloom language and only Bloom programmers can use it.

Bassil (2012) proposed testing architecture for services oriented architecture based applications (SOA). SOA Applications could be distributed applications running in heterogeneous environment thus testing under different deployment configurations must be conducted. Proposed testing architecture supports testing of multiple web services. This parallel execution of test cases increases throughput and reduces time. Chan et al. (2007) proposed framework for testing SOA applications. This study is extension of Chan et al. (2005). Proposed framework uses metamorphic testing approach. Metamorphic testing employs mathematical relations called metamorphic relations to conduct testing. It is observed that using proposed approach with 16% less efforts 13% more defects were identified.

Snelick et al. (2009) proposed framework for testing distributed healthcare applications. In healthcare industry application are acquired from many vendors thus conformance and interpretability testing is required and this study is aimed to address these two issues. Hanawa et al. (2010) proposed cloud based software testing environment for parallel and distributed applications. Study is aimed to address defect reproducibility and fault tolerance. Wu et al. (2011) is cloud based performance and compatibility testing framework for web

applications. Hengliang et al. (2013) proposed cloud based testing platform for increasing testing efficiency. Tómasson and Neukirchen (2013) have proposed TTCN and Jata based framework for testing cloud applications. During experimentation instead of deploying PTC on worker nodes, stubs were used to return results to MTC because delaying actual PTC on worker nodes is proved very difficult. Stepien et al. (2008) proposed TTCN based framework for testing web applications. Key testing challenges identified are features rich web interface, client side scripting and reusable component based server side application. The framework uses specification-based approach and test agents. Test specification approach provides many advantages over other techniques such as test cases reusability at different level of abstractions and testing. But to obtain executable test cases an adoption layer must be implemented. TTCN-3 demands higher levels of skills from test engineers thus become major drawback of this framework.

Mišic et al. (1998) proposed a framework for testing distributed multimedia software systems. Study identified stringent timing and synchronization requirements for testing and categories errors and test SUT for three categories i.e. timing errors, synchronization errors and functional errors.

Lübke et al. (2014) is large scale distributed testing platform composed of Test Systems (TS) to emulate the behavior of SUT. TS run Test Nodes Modules to initiate and control SUT. To enable TNM to control SUT, NESSEE must be integrated into development process and incorporate IPC mechanism. Test cases are executed by script engine built in NESSEE server and TNM. This framework also includes a network emulator called Degrader to test environment and provides a generic architecture for scalability tests of client/server-based systems.

Testing distributed systems is difficult due to challenges such as locks, time outs, controllability, observability, and synchronization problems. Testing process of distributed system must check if the output events have been observed along with the time when event occurred. Azzouzi et al. (2015) proposed a multi-agent based distributed testing architecture. This study illustrates how to overcome these problems by using a distributed testing method including timing constraints. This study focus on temporal properties of distributed systems, which specify the time required to exchange messages among different components of the distributed test application. An algorithm has been developed to overcome problems of observation, coordination and synchronization which generate Timing Local Test Sequences for each tester.

Mirshokraie et al. (2015) proposed mutation testing technique for JavaScript based web applications. Mutation testing technique has a higher computational cost involved to execute test suite against a large set of generated mutants. In this study a metric named FunctionRank is proposed along with an algorithm to select variables and branches for mutation. A static and dynamic analysis technique is developed to guide the mutation generation process towards parts of the code that are more likely to influence the program's output.

Hierons (2015) worked on developing complete test suites for distributed testing. This study proposes development of cm-complete test suites. This study is based on hypothesis that SUT has no more than m states and a cm-complete test suite achieves as much as is possible given that testing should be controllable.

Distributed system lacks determinism thus reliability in the network may cause applications defects. These defects are hard to identify and reproduce therefore hence making it difficult to remove these defects. MapReduce service is one such example of distributed systems. MapReduce enable processing and storage of large amount of data, Marynowski et al. (2015) developed an technique to test fault tolerance of MapReduce. In this study a method is developed to generate set of fault cases using Petri Nets and framework for automated execution of these fault cases in a distributed system.

Critical and distributed component-based systems are gaining increasingly population. Such system cannot be stopped for maintenance and up gradation. Therefore, runtime evolution / dynamic adaptation are more and more required. It is used frequently. This is done by dynamically modifying the software architecture or by modifying its behavior which increases risk of introducing defects. Lahami et al. (2016) proposed standard-based and resource aware runtime testing framework for adaptable and distributed systems to overcome above mentioned problem. Proposed framework carry out distributed testing at runtime, preventing interference between test processes and business processes.

## 3. DisTest: A distributed testing model

Proposed model DisTest is capable to work in any environment which supports Java. DisTest uses commercial-off-the-shelf test automation and defect tracking tools. DisTest is independent of vender, specific tools, hardware or operating system. This enables DisTest to reduce upfront cost of acquiring any specific test automation or defect tracking tool. By executing an end to end flow of functionalities, DisTest test scenarios provide a bigger picture of application status to test engineer. Test engineer can observer stable and unstable functionalities hence increases confidence on quality of product. By identifying unstable functionalities, proposed model enables test engineer to put more effort testing unstable functionalities. DisTest executes part scenarios in parallel therefore testing time could be reduced. This parallel execution of part scenarios also results in better control over testing activities and application status since test engineer can executes desired test cases in parallel at different nodes and can uncover system level defects. DisTest

Model consist of three layers. These three layers are Test Oracle, Testing Technique, and Test Execution Engine.

## 4. Test oracle

Test Oracle in conventional terms is used to verify the result of test cases and for that purpose it could use a heuristic or a uses case(Hierons, 2012). But in this study we have extended its definition that it also act as a repository which holds all testing related artifacts (Lübke et al., 2014). These artifacts include tractability matrices, test scenarios, test cases, test scripts, and defect logs.

## 5. Scenario based testing technique

DisTest uses scenario-based testing technique. In proposed model test engineer could control and monitor state of application on different nodes thus can identify system level defects. DisTest is flexible and can be employed at any testing level e.g. component, integration and system level. In case where some components or modules of SUT are in development phase DisTest can be used with necessary stubs and drivers to fill in the missing parameters (Meszaros, 2007).

## 6. DistTest test execution engine

Test Execution Engine is responsible to execute test scenarios in a distributed test environment. DisTest model is depicted in Fig. 3. It consists of DisTest Server, DisTest Client and Communication Service and. DisTest Server execute on server machine and controls the test execution. DisTest Server consist of three components i.e. Synch Manager, Scenario Explorer and Result Analyzer.

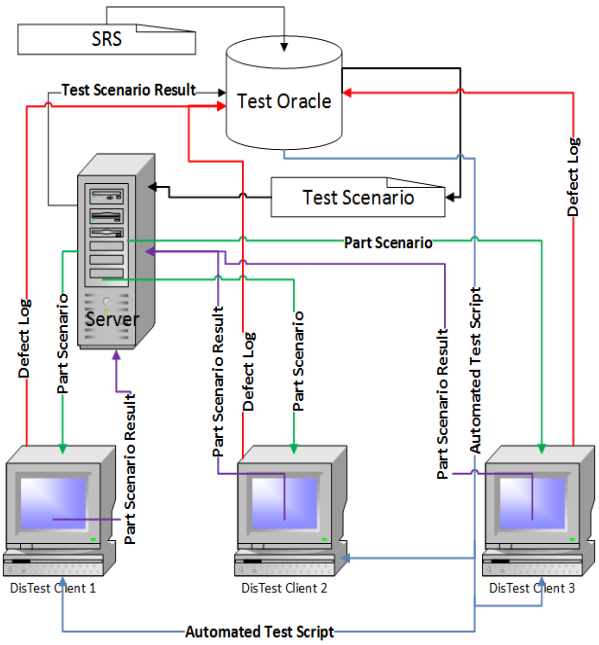


**Fig. 3:** Proposed model – DisTest

DisTest Remote Client is a lightweight component which executes on a client machine. It has four components which are Synch Client, Test Execution Engine, Result Cliper and COTS Test Automation Tool. Test Execution Engine controls the execution of part test scenario while test case execution is responsibility of configure test automation tool. Result clipper collects result of every executed test case. On completion of part scenario it compiles a part test report and synch client component sends this report to DisTest Server. DisTest Remote client is developed in an extendable manner and new commands can be added easily. DisTest Remote Client can be configured with any COATS test automation tool or proprietary test automation tool.

Communication Service provides three main functionalities: Security, Service Discovery, and Load Balancing. Security means establishing secure and encrypted data transfer channel among all components of the system. When a node request part scenario, its request may be encrypted before appearing on communication channel. Transfer of Test Script, Part Scenario, and Part Scenario Result could also be encrypted and decrypted. This is done by the communication service.

In distributed application, components are deployed at different nodes which can be configured in such a way that more than one component are replicated on few nodes. Service Discovery i.e. identifying which component of system under test is deployed at what node, is responsibility of this component. There could be a situation where need for load balancing is felt e.g. a remote client executing a part test scenario unexpectedly gets disconnected from network. In such case part scenario need to be moved to some other node of the system. Part scenario can only be moved to a node where required components are already deployed. This information will be obtained from service discovery component. At present Communication Service and Network Topologies related to proposed solution which caters for Security, Service Discover and Load balancing is assumed to be in place and hence is out of scope of this study.

## 7. Proposed model validation

Model validation is a critical task. Two key points sufficient accuracy, and built for a specific purpose have been identified for model validation. No model can be 100% accurate therefore model validation aims to access that a model is sufficiently accurate to achieve the purpose for which it is built (Robinson, 1997, 2014). Researchers have developed many methods for simulation model validation such as conceptual model validation, Black-box validation, Experimentation validation, white-box validation, and Solution validation.

To validate proposed model, we have used case study based black box validation strategy. This technique is used to check accuracy of, overall or macro operation of the model. In this technique primary concern is to validate whether a model

provides a sufficiently accurate representation of the real world system to meet the objectives of the simulation study. For designed case study test cases, automated test scripts and test scenarios are developed and simulated proposed model on case study. Furthermore results are collected and compared our work with other similar studies e.g. NESSE (Lübke et al., 2014).

## 8. Case study

This experimentation is not intended to perform real life testing and uncover defects in application under test. Scope of the case study is limited to demonstrate operational accuracy of proposed model. Modern applications are complex and may be composed of various user roles and components. Access to functionalities or components is subjected to roles based privileges. In this case study we have evaluated proposed model to access its ability when employed to test complex modern applications. We have used Alfresco Enterprise Collaboration and Content Management System as system under test. This case study evaluates the accuracy of model by executing a scenario which is based on test cases having different users/roles found in modern enterprise applications. Test scenario is based on three user roles i.e. Alfresco Administrator, Alfresco Site Administrator and Alfresco End User. Depending on security privilege / permissions, each role has access to different set of functionalities and for that we have developed several test cases for each user role. These test cases are listed in Table 1 and test scenarios designed for Alfresco ECM is shown in Table 2.

**Table 1:** Test cases for alfresco enterprise collaboration and content management system

| Test Case ID | Test Case Description | Module |
|---|---|---|
| AD_01 | Verify that Alfresco admin can create a user. | Alfresco Admin |
| AD_02 | Verify that Alfresco admin can disable a user. | Alfresco Admin |
| AD_03 | Verify that Alfresco admin can delete a user. | Alfresco Admin |
| SA_01 | Verify that Alfresco admin create a collaboration sharing site. | Alfresco Site Admin |
| SA_02 | Verify that Alfresco admin can delete a collaboration sharing site. | Alfresco Site Admin |
| AEU_01 | Verify that Alfresco end user can join a public site. | Alfresco End User |
| AEU_02 | Verify that Alfresco end user can leave a public site. | Alfresco End User |

**Table 2:** Test Scenarios for Alfresco ECM

| S ID | TC | TC ID | Result | Description |
|---|---|---|---|---|
| 01 | 07 | AD_01,AD_02,AD_03, SA_01,SA_02,AEU_01,AEU_02 | Pass | Validate the core functionality of all roles of Alfresco ECM. |

The above mentioned test scenario is executed with DisTest Test Execution Engine and result is shown in Fig. 4. Result analyzer generated dual axis graph for each node. On x-axis name of test cases are provided. On left Y-axis number of iterations each test case executed is listed. On right y-axis a line is draw to depict fail thresholds of test cases are provided. Bars in red color depict iteration with status Pass and blue bars depict number of times a test case Failed.

NESSEE described in study Lübke et al. (2014) has used video conferencing application while we have used Alfresco Enterprise Collaboration and Content Management, but we can compare our model with NESSEE. The biggest advantage of NESSEE is that it has a component named Network Degrader to emulate large networks. While biggest demerit of NESSEE is that, to enable it to control SUT, application developers must integrate NESSEE into development process and incorporate inter process communication mechanism in application. This also means that NESSEE cannot be used with components acquired from third party vendors.

## 9. Analysis

Fig. 5 shows overall picture of ongoing testing activities. This is a dual axis graph. On x-axis time and on y-axis scenarios are depicted. Scenarios with status pass are depicted with 1 and failed scenarios are depicted with 0. While a line graph is showing start time of each scenario.

| | |
|---|---|
| *Scenario ID:* | 1 |
| *Scenario Name:* | Alfresco ECM Core functionality |
| *Scenario Status:* | **Pass** |
| *Scenario Start Time:* | 09/17/2015 13:00:00 |
| *Scenario End Time:* | 09/17/2015 14:18:54 |
| **Part Scenario Result** | |
| Node IP: | 192.168.102.30 |
| Start Time: | 09/17/2015 13:00:00 |
| End Time: | 09/17/2015 13:18:54 |
| Status | **Pass** |

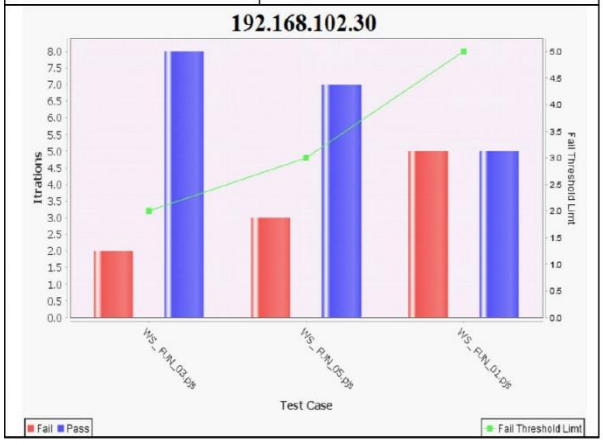


**Fig. 4:** Alfresco ECM Test Scenario Result

With the help of this graph test engineer can monitor the execution of test scenarios with respect to time as well as test engineer can observer the overall stability of application.

Fig. 6 shows scenario level testing activities. This is also a dual axis graph. On x-axis client id is provided while on y-axis part scenario id is given. Graph in blue line is depicting part scenario distribution to the client nodes.

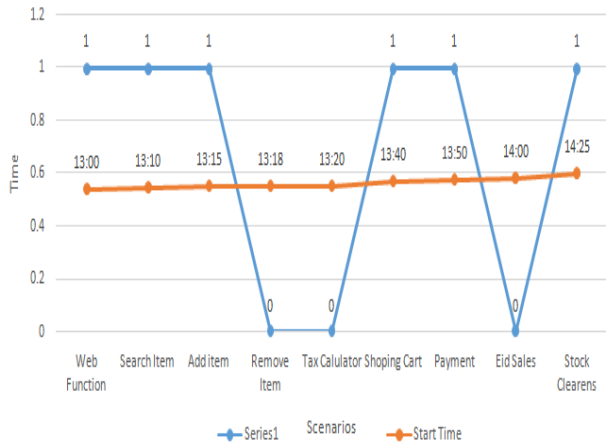


**Fig. 5:** Visibility graph of testing activities

On analyzing the graph test engineer can observe that part scenario 1, 2, and 3 are assigned to client id 1, 2, and 3, respectively. Client ID 3 has executed assigned part scenario. On completing of part scenario in 3 it submitted the result and quires server for part scenario waiting for execution. Server has assigned client id 3 part scenario id 4 for execution.

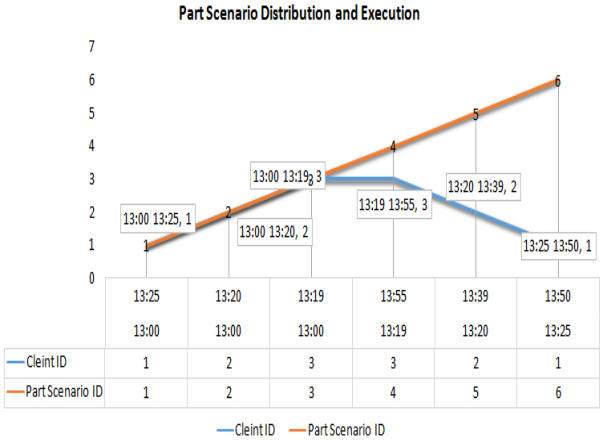


**Fig. 6:** Part scenario execution

This is to achieve batter resource utilization. After executing a part scenario client node submit result to server and if on server a part scenario exist and waiting for execution it is assigned to that node. Client id and part scenario along with execution time i.e. start time and completion time of part scenarios are shown in Fig. 6.

In Fig. 7 it shows control on over all application status. Part scenarios are executing in controlled manner thus initiation of functionality, number of testing nodes and application status is in control of test engineer.

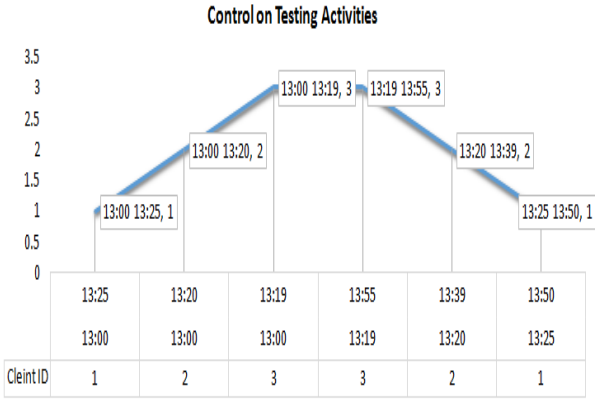


**Fig. 7:** Control of testing activities

If scenario fails, location, time and reason for failure can easily be identified. This also makes defect reproduction easy as test engineer know overall picture of testing activities.

## 10. Conclusion

DisTest uses scenario-based testing technique in distributed fashion and increases stakeholder's participation in system development and many design level defects can be uncovered at initial stage of software development. Test prioritization is a complex task and requires time and efforts. Test scenarios are based on user's priorities and expectation's from systems functionalities and significantly reduces efforts and time required for test prioritization and selection. Experimental results shows DisTest provide better control and visibility during testing at any specific stage which provides a complete view of software application regarding its maturity and stability. DisTest support interoperability with any COTS test automation tools and defect tracking tools. It also reduces upfront cost of acquiring any specific test automation or defect tracking tool. DisTest also supports portability since it is developed in Java and can be deployed in any supports Java environment. Results are easy to understand and status of scenario is updated in Test Oracle thus no specific tool is required for recording results of test scenarios.

## References

Alvaro P, Hutchinson A, Conway N, Marczak WR and Hellerstein JM (2012). BloomUnit: Declarative testing for distributed programs. In the Proceedings of the Fifth International Workshop on Testing Database Systems (DBTest '12). Scottsdale, Arizona. https://doi.org/10.1145/2304510.2304512

Azzouzi S, Benattou M and Charaf MEH (2015). A temporal agent based approach for testing open distributed systems. Computer Standards and Interfaces, 40: 23-33.

Bassil Y (2012). Distributed, cross-platform, and regression testing architecture for service-oriented architecture. Advances in Computer Science and its Applications, 1(1): 9-15.

Chan WK, Cheung SC and Leung KR (2005). Towards a metamorphic testing methodology for service-oriented software applications. In the IEEE Fifth International Conference on Quality Software (QSIC'05): 470-476. https://doi.org/10.1109/QSIC.2005.67

Chan WK, Cheung SC and Leung KR (2007). A metamorphic testing approach for online testing of service-oriented software applications. International Journal of Web Services Research, 4(2): 61-81.

Desikan S (2006). Software testing: principles and practice. Pearson Education India.

Dhavachelvan P, Uma GV and Venkatachalapathy VSK (2006). A new approach in development of distributed framework for automated software testing using agents. Knowledge-Based Systems, 19(4): 235-247.

Gupta D, Vishwanath KV, McNett M, Vahdat A, Yocum K, Snoeren A and Voelker GM (2011). DieCast: Testing distributed systems with an accurate scale model. ACM Transactions on Computer Systems (TOCS), 29(2): 1-15.

Gupta R and Bajpai N (2014). A keyword-driven tool for testing web applications (KeyDriver). IEEE Potentials, 33(5): 35-42.

Haeberlen A, Mislove A and Druschel P (2005). Glacier: Highly durable, decentralized storage despite massive correlated failures. In the Proceedings of the 2nd conference on Symposium on Networked Systems Design and Implementation. USENIX Association, 2:143-158.

Hanawa T, Banzai T, Koizumi H, Kanbayashi R, Imada T and Sato M (2010). Large-scale software testing environment using cloud computing technology for dependable parallel and distributed systems. In the IEEE Third International Conference on Software Testing, Verification, and Validation Workshops (ICSTW): 428-433. https://doi.org/10.1109/ICSTW.2010.59

Hengliang S, Changwei Z, Tao H and Yongsheng D (2013). Research on distributed software testing platform based on cloud resource. International Journal of Computer Science and Engineering Survey, 4(2): 17-25.

Hierons RM (2012). Oracles for distributed testing. IEEE Transactions on Software Engineering, 38(3): 629-641.

Hierons RM (2015). Generating complete controllable test suites for distributed testing. IEEE Transactions on Software Engineering, 41(3): 279-293.

Kaner C (2003). The power of "What If" and nine ways to fuel your imagination: Cem Kaner on scenario testing. Software Testing and Quality Engineering, 5: 16-22.

Lahami M, Krichen M and Jmaiel M (2016). Safe and efficient runtime testing framework applied in dynamic and distributed systems. Science of Computer Programming, 122: 1-28.

Lübke R, Schuster D and Schill A (2014). NESSEE: An in-house test platform for large scale tests of multimedia applications including network behavior. In the Springer International Conference on Testbeds and Research Infrastructures: 229-238. https:/doi.org/10.1007/978-3-319-13326-3_22

Marynowski JE, Santin AO and Pimentel AR (2015). Method for testing the fault tolerance of MapReduce frameworks. Computer Networks, 86: 1-13.

Meszaros G (2007). xUnit test patterns: Refactoring test code. Pearson Education. New Jersey, USA.

Mirshokraie S, Mesbah A and Pattabiraman K (2015). Guided mutation testing for javascript web applications. IEEE Transactions on Software Engineering, 41(5): 429-444.

Mišic VB, Chanson ST and Cheung SC (1998). Towards a framework for testing distributed multimedia software systems. In the IEEE Computer Society Proceedings of the International Symposium on Software Engineering for Parallel and Distributed Systems.

Mogul JC (2006). Emergent (mis) behavior vs. complex software systems. In ACM SIGOPS Operating Systems Review, ACM, 40(4): 293-304.

Paydar S and Kahani M (2011). An agent-based framework for automated testing of web-based systems. Journal of Software Engineering and Applications, 4(02): 86-94.

Robinson S (1997). Simulation model verification and validation: increasing the users' confidence. In the IEEE Computer Society Proceedings of the 29th Conference on Winter Simulation: 53-59.

Robinson S (2014). Simulation: the practice of model development and use. Palgrave Macmillan, Basingstoke, UK.

Snelick R, Gebase L and O'Brien G (2009). A framework for testing distributed healthcare applications. In the International Conference on Software Engineering Research and Practice (SERP'09), Las Vegas, USA.

Stepien B, Peyton L and Xiong P (2008). Framework testing of web applications using TTCN-3. International Journal on Software Tools for Technology Transfer, 10(4): 371-381.

Tómasson H (2011). Distributed Testing of Cloud Applications Using the Jata Test Framework. In

the Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies (NordiCloud '13). https://doi.org/10.1145/2513534.2513540

Tómasson H and Neukirchen H (2013). Distributed testing of cloud computing applications using the TTCN-3-based Jata test framework. In the Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies (NordiCloud '13), Oslo, Norway. https://doi.org/10.1145/2513534.2513540

Wu J, Wang C, Liu Y and Zhang L (2011). Agaric—A hybrid cloud based testing platform. In the IEEE International Conference on Cloud and Service Computing (CSC): 87-94. https://doi.org/10.1109/CSC.2011.6138558.