On Test Patterns for Cloud Applications

Sidra Siddiqui
Department of Software Engineering
Bahria University, Islamabad, Pakistan
ssqr1790@gmail.com

Tamim Ahmed Khan
Department of Software Engineering
Bahria University, Islamabad, Pakistan
tamim@bui.edu.pk

***Abstract*—Software testing is an important aspect for the quality of software. Different levels of experience, types of application and needs enforce differences in performing testing activity for same features of applications. Testing as an activity within a development house usually includes testing of recurrent situation e.g., testing of security features etc. Development of a homogeneous test ground requires considering information regarding the structure of real world scenarios.**

We propose a test pattern based technique which supports identification of test cases on the basis of specification and domain analysis. The proposed technique provides support for Test Driven Development (TDD) and Test Last Development (TLD). We provide test patterns for testing cloud applications where we study what features an application would bear and we propose what test cases must exist in the test suite for that application. For the purpose of evaluation, we consider threats to cloud applications and discuss test patterns.

Keywords—Test driven development; test last development; software testing; pattern based testing.

I. INTRODUCTION

Software testing is a complex task in software development process. In both TDD (Test Driven Development) and TLD (Test Last Development), each new update in development procedure of software introduces new challenges. Pattern based testing models recurrent situations in testing such as testing access control and provides a structure which is homogeneous to support testing for these situations.

Patterns are packages of reusable knowledge that can be used as common efforts to solve problem supporting reusability [13]. More specifically, patterns are useful to define something that is recurrent, describe repetitive behavior and their associated solution [12]. In terms of testing, patterns are strategies used to conduct testing which can be combined with existing patterns. They are test templates or test patterns that have context, intention, situation, action and some results in the form of test case development, or test case execution.

Pattern based testing previously proposed considers design, security, GUI, and Defects in [4] [13] [15]. However, all of these approaches focus on testing of how effectively the pattern has been implemented or detecting recurrent situation. The issue regarding standardization of testing need to be elaborated since we require an experience independent testing approach so that we are able to consider domain information as well. Such a testing requires domain analysis with respect to preset and provided templates, and selects all those that are applicable. We propose an approach which extends this concept where we provide detailed description of pattern structure and how it supports testing in practical. For the purpose of brevity, we consider Cloud vulnerability study that provides a good collection of threats in [23] and it presents a detailed taxonomy of such vulnerabilities. We connect threats to test patterns and use a case study for demonstrating usefulness of our approach. This way, we provide support for TDD as well as TLD and outlines how we can automate testing using test patterns.

The paper is organized as follows. We present our proposal in Section II and present our evaluation in Section III. Section IV is dedicated to case study. Sections V consists of related work. We; finally, present our conclusion and outlook in Section VI.

II. OUR PROPOSAL

We consider cloud vulnerability taxonomy as proposed in [23] and propose a general structure or test patterns. Our approach comprises of three parts: analysis, processing and definition. In the analysis stage, we analyze specifications and domain properties to which application belongs. The aim is to identify situation and the output of this stage is identification of positive situation and negative situation of use. Here we study how users respond to the specification. The next stage is the processing where the input to this stage is data regarding situation and result of processing is identification of associated risks. These risks are grouped in two categories: Sub Category and main category (low level and high level risks). Processing stage also helps in identification of low level action which is performed to achieve a situation of use. Finally, we perform definition stage where actions and risk are combines together for generation of test pattern. The other output of this stage is the identification of application properties which software application must have for selecting a test pattern to generate test cases. We call these properties as application subject. The overall approach is shown in Fig 1. Each pattern has three basic elements - Situation, Action, Success situation for the pattern implementation. Template for each pattern must contain the following attributes:

NAME: To identify test pattern.

TYPE Category: Defines if the test is based on event, trend or behavior.

978-1-5090-5300-1/16 $31.00 © 2016 IEEE
DOI 10.1109/FIT.2016.17

Defect Type: Indicates type of defects; after effects of attacks are described as defects types.

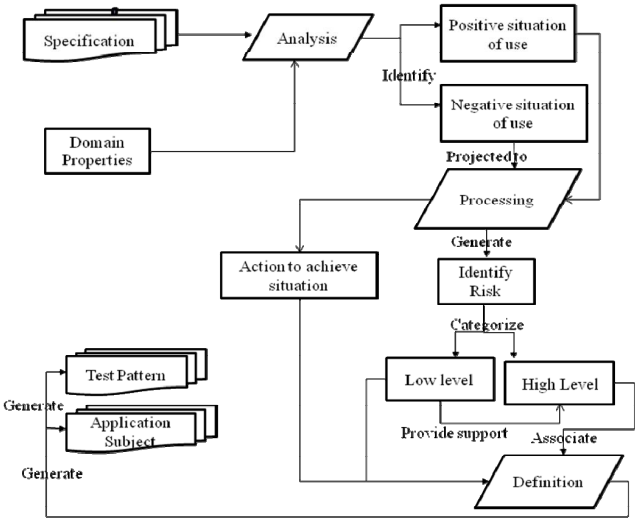

Fig. 1. Approach Flow

Test Pattern Type: It includes type of testing and the relationship of test with the development level.
GOAL: What testers intend to achieve from this form of testing
Sub category: Sub category Threats are the low level attacks which cause the threats of the associated patterns. These attacks are the basic reason which lead to the pattern threat
Situation: Actual issue to be tested (i.e. situation of use).
Target: Issues that are expected to be revealed, target system part and system responses.
Action: Sequence of action need in order to perform the test. These actions are base for each subcategory in the form of ACTION 1 and ACTION 2. Otherwise if the actions for subcategory interact with each other in the joint fashion they are consider as one set of consolidated ACTION attribute.
Success criteria: Arrive at required attack.

We study 37 threats [23] related to Services Oriented Architecture for development of test patterns. In these pattern we only consider those that can be duplicated in lab environments. The pattern which we do not consider include malicious insider, privacy breach, natural disaster, side channel attack, reliability of data calculation carried out, misuse of infrastructure, hardware theft, migration of virtual machine, breach of contractual rights, sanction due to political issues, and unknown risk profile. Our considered threats possess describable properties which express their occurrences or non-occurrences and have manageable cost of testing and visible outputs. These threats are further subdivided into main and sub categories of Threats. Sub category threats are low level attacks which cause threats of associated patterns and they are also termed as low level risks. These attacks include examples such as Denial of Service (DoS) attacks, Session hijacking, Resource contention, Data interruption, Target modification, Access limit and trust level on shared VM environment, Hypervisor Compromises, Insecure Interfaces, Shared Technology Issue, and Discontinuity of external resources, Virtual machine Image and controlled access. These threats cover a broad spectrum and are associated with host, platform, application, infrastructure and administration. Threats remain the same; however, they have some variation in parameter of effect at different levels. Examples include DoS attack at network level will have a similar effect at application level and Session hijacking at application have impact at the data level and it also depends on data level issues for its successful completion. We present our categories, sub-categories, positive and negative situations of use in Table 1.

TABLE I. CATEGORIES, SUB CATEGORIES, POSTIVE/NEGATIVE SITUATIONS OF USE

Category	Sub-Cat	Positive/Negative use
Data Security testing, Data interruption	SQL injection, updating without back up	▪ Legal Access of required data to avail system service. ▪ Access data for illegal use by exploiting system services. ▪ Modify data to reflect new update.
Volume limitation	DOS Attacks, Connection flooding, Resource usage max limits.	▪ Multiple users accessing system. ▪ Using unavoidable service by multiple users at the same time. ▪ Generating multiple request to effect system integrity ▪ Authorized user (un)able to access system.
Target modification (intercepting and Modifying message)	XSS Attack, Redirection, invalidated input with Html encoding disable.	▪ Participate in multi user activity ▪ Redirecting system to a newer version of system without info. ▪ Redirecting system to effect integrity of system, trust of users, posting infected reply to infect associated users.
Session hijacking	Brute force profile login, Access after session expiry, Cookie data access	▪ Authentication mechanism. ▪ Keeping session information, transaction tracking, keeping track of states during long interaction. ▪ Stealing user information to hijack user account, to manipulate user resources and to generate illegal act on behavior of user.
Access limit and trust level on shared VM environment	Resource usage limit functionality, system response to unauthorized user for critical data,	▪ Different user rights and access levels, sharing file, data and information on common resources and shared files, Multiple files and coordinated users and their access rights. ▪ Data access levels with different access controls. ▪ User role enforcement management. ▪ Authorized user with unauthorized data access for modifying, malicious user can manipulate shared data.
Hypervisor compromises	Lack of Intrusion detection and prevention, Rootkit, Data integrity verification is absent.	▪ Running plug-in on servers or server driven installation of software which need antivirus independent environment. ▪ Multiple format files uploading and sever file corruption or virus attack infect the other guest user. ▪ An admin access with a rootkit malware breaching the security and run malicious code on root system.

		▪ Intrusion to manipulate all files on server and client system.
Insecure Interfaces	Visible passwords, Lack of data flushing after each use, Lack of control over API's use and data access limit, lack of logging mechanism for each usage.	▪ Login interface to access profiles. ▪ Keeping track of information for specific time duration to improve user experience. ▪ Avoiding time out and data refresh on small interval for better user interactions. ▪ Visible password to help user to see what they are typing. ▪ Third party interaction for payments or SOA architectures. ▪ Unencrypted password can lead to access with unauthorized intention, Non Flushing of data fields lead to unwanted access to system. Over provision of system control through the API's to the user, compromises the system security by allowing access to critical data. ▪ Lack of logging mechanism, unwanted third party access.
Discontinuity of external resources	Connection lost before saving information, Data server connection lost during storage, Incomplete transaction.	▪ Back up storage and long transactions. ▪ Center data point. ▪ A legal user is trying to store information but suddenly internet connection is lost. ▪ A user has used the system success fully however, the data center fail to store the data.

We share data security testing/data interruption test pattern. The attack can be achieved by an insecure data access or by manipulating query through SQL injection. SQL injection facilitates the purpose of invalid unauthorized access and manipulation of data. A successful execution of SQL injection on to SOA system indicates that it is vulnerable to such threats. The proposed data security testing or data interruption pattern contains subcategories that comprises of SQL injection and updating data without backup. The after effects of these attacks are described as defect types. The basic problem with the system which can successfully be attacked is the lack of system surveillance towards data manipulation. The system possesses a weak data manipulation control channel which leads to the success of such attacks. Table II shows the pattern.

TABLE II. TEST PATTERN DATA SECURITY

Name	Data Security issue testing or Data interruption
Type/category	Behavior
Defect types	Data mishandling for malicious manipulation/unauthorized access
Test pattern type	Threats from real system scenario are applied on system as test case to check its resistance.
GOAL	- To check the system response on a data security attack. - To minimize future risks of data security when system is deployed. - Test the system failure limit on data associated attacks.
Sub category	SQL injection, updating without back up
Situation	Unauthorized access to data to cause damage to system security. Minimal system response on data manipulation may leads to unsaved access or manipulation without back up causing damage to system. Allowing user to query by an open access point can cause injection.
Target	-To uncover system failure in test by applying threats such as SQL injection and checking system response. -Test system by trying to update without back up.
Action	**1. Access URL.** 2. **Pre-Condition:** There must be some data in the database. **3. Choose Input Field** **4. Choose the database query.** 5. **Input:** Run a data retrieval query. 6. **Get the system response:** Output data/ alert. **7. Compare expected and actual response.**
Success criteria	-Data is illegally accessed through SQL injection, deleted from table. -Data changed without backup request, data lost.

In order to test our pattern, we run the test case which is shown in Table III. We first access the system, select required target field and then we choose required data query. We first choose a valid select query to get data status and check system response.

TABLE III. GENERAL TEST CASE FOR PATTERN - DATA SECURITY

Test case	Example 1
Input 1	Run Select * table 1
Output: Data	Data table
Input 2: Data	Run Malicious Select Query
Output: response	Generate no alert
Output: Data	System Critical Data
Expected Output	Data
Actual Output	Data
Success scenario	Expected = Actual

We propose a total of 8 patterns in total covering 37 threats studied from [23]. We do this for the purpose of brevity and we submit a complete list of these test patterns[1]. Our list of test patterns includes data security, data interruption, volume limitation, target modification (intercepting and, modifying message), session hijacking, access limit and trust level on shared VM environment, hypervisor compromises, insecure interfaces, and discontinuity of external resources which encompass all 37 threats and categories shared in [23].

III. EVALUATION

In order to demonstrate the applicability of our proposal, we conduct case studies of real time projects. We analyze system requirement specifications and extract features of each project after analysis. The features define requirements, management, abilities and core properties. These properties include attributes such as, authentication, managing multi users, multiple file, feedback system, back up storage and reboot mechanisms. We also perform test pattern analysis along with the feature analysis. This helps us to match identified features with the test pattern application subject areas and the domain

[1] http:www.se.bui.edu.pk/wp-content/uploads/2015/01/patterns.pdf

properties. Merging of these two information support the defining process of general test cases. To design specific test cases SRS provides data as shown in Fig 2.

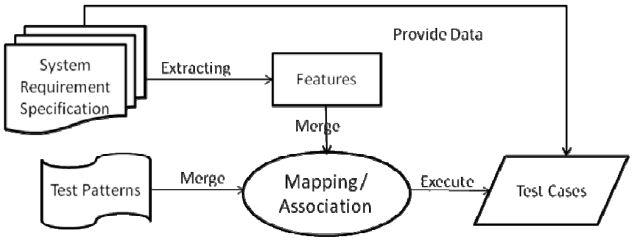

Fig. 2. Evaluation process diagram

We choose Facebook, Dropbox and Moodle. All of these systems require user sign in mechanism, have multiple users, manages large amount of data, manages multiple files, have shared forums and multiple active user and action active at one time. They support many to many relationships between services provided and the users. Considering the issue of privacy, we only run those test cases which Do not affect any data. All of these popular systems are designed with pre-mitigation against the basic attacks like SQL injection, XSS attack and DoS attack. Moreover, these websites are designed to properly manage role between the users. We divide testing activity for these systems under test (SUT) into two groups A and B. A consist of Facebook and Dropbox as SUT and B consist of Moodle as SUT.

A. Testing Group A

A system which Does not respond positively on a negative test case (i.e. an attack) is actually previously developed with immunity to that attack. We analyze SUT and identify features, services which it provides. The list contains Login, Manage multiple users, Shared Forums, Initiate multi user discussion, Upload/ Download file, Upload /Download file on shared forums, Provide Server storage, and Edit personal profile data.

As shown in the approach flow shown in Fig 1, the output of analysis is the identification of situations of use. We identify expected associated threats which are Password stealing, Unauthorized access, Profile hijacking, Manipulation of critical data by unauthorized access, Access limitation and trust level on shared VM environment, Un awareness of change, Malicious content propagation, Data interception, Volume limitation, Target modification, Insecure interface, Hypervisor compromising and Disconnection of external resource to terminate transaction.

We map expected threats and services to Table I where mapping identifies associated categories, subcategories and associated situation of (mis-)use. The associated patterns are; insecure Interfaces, discontinuity of external resources, access limit and trust level on shared VM environment, and hypervisor compromises. We apply these patterns on our SUT and run specific test cases. Here we only show one example of test case in Table IV for the purpose of brevity. The rest of the test cases can be generated by using out test patterns proposed general test cases. The overview of the case study is shown in the Table VI.

TABLE IV. EXAMPLE TEST CASE

Test case	Example 1
Reference to general test case	Table 20
Input 1	Select the file to upload
Action	Disconnect the internet
Actual Output	File which was previously uploading give indication to user to retry.
Expected output	File gets lost on next time when system reboots. System does not provide support for previous state of user file upload.
Success criteria	Actual output=Expected

B. Testing Group B

Moodle manages multiple users with different roles. It is used for managing content sharing and management in educational institutes. Moreover, it provides services to manage concurrent activities such as on-line testing and assessments. We implement our approach and follow steps as shown in Fig 1. We analyze system and identify services which it provides that includes Login, Manage multiple users e.g., teachers, students, Shard Forums, Manage Courses, Simultaneous submission and assessments, Initiate multi user discussion, Upload/ Download file, Upload /Download file on shared forums, Provide Server storage, Edit user profile data, Feedback and Notify user on new uploads.

We process the requirements and understand the domain area. As shown in the approach flow shown in Figure I the output of analysis is the identification of situation of use. We have identified the expected associated threats which are listed as Unauthorized access, Profile hijacking, Brute force access, Lack of Feedback, Manipulation of critical data by unauthorized access, Access limitation and trust level on shared VM environment, Un awareness of critical data manipulation, Malicious content propagation, Data interception, Volume limitation, Target modification, Insecure interface, Hypervisor compromising, and Disconnection of external resource to terminate transaction.

We map expected threats and services considering Table I and we find associated patterns which are - data security issue testing or data interruption, target manipulation, session hijacking, insecure interfaces, discontinuity of external resources, access limit and trust level on shared VM environment, hypervisor compromises. We apply these patterns on the system and run specific test cases. Here we only show one example test case in Table V for the purpose of brevity. The overview of case study is shown in Table VI.

TABLE V. EXAMPLE TEST CASE

Test case	Example 1
Input 1	Run Select * table 1
Output: Data	Data table
Input 2	Run Select like clause query with addition

	information ”q ; Delete from table –“
Output: System response	Generate no alert
Output: Data	No data
Input 3:	Run select * table1
Output: Data	No data table deleted
Expected Output	No data
Actual Output	No Data
Success scenario	Expected = Actual

IV. Related Work

UMLsec extension for system development and modeling were proposed in [6]. New stereotypes of UML were identified by the author to address security situations. Comparison of misuse cases with other techniques is proposed in [11]. Likewise, misuse case support eliciting of security requirement is proposed in [10].

TABLE VI. Testing using test patterns result

System	Test Pattern	Negative responses/ Positive Responses	Available mitigation
Facebook	Insecure Interfaces, discontinuity of external resources, access limit and trust level on shared VM environment, hypervisor compromises	System provides 80% of negative response on the positive test case. Corrupted file gets propagated infected clients.	Encrypted password. Feedback procedure. State management. User Profile, access right management. Different level of security for different users. Role and responsibility divisions.
Drobox	Insecure Interfaces, discontinuity of external resources, access limit and trust level on shared VM environment, hypervisor compromises	System provides 75% of negative response on the positive test case. Corrupted file gets propagated infected clients.	Encrypted password. Feedback procedure. State management. User Profile, access right management. Back up support. Offline upload.
Moodle	Data Security issue testing or Data interruption, Target manipulation, Session hijacking, Insecure Interfaces, discontinuity of external resources, access limit and trust level on shared VM environment, hypervisor compromises	System provides 60% of negative response on the positive test case. The system shows a positive response to the brute force access to profile test case. System show negative response on the positive test case of feedback on critical data manipulation. The corrupted file can infect the other user.	Encrypted password, managing multiple user with different access rights. Managing multiple file formats, users and activities independently.

These security requirements are supported by security patterns proposed by several researchers. A classification of these pattern was proposed in [7]. The authors described that the use of patterns at design level is difficult because detailed information of vulnerabilities is limited. They argued that identification of these vulnerabilities at early level can bridge the gap of security pattern selection and help in testing the software [5]. Exploiting security or attacker pattern can help in identification of expected testing. A methodology of modeling attacker pattern and then using them for testing was proposed in [5]. XSS attacker pattern is discussed for testing in [18]. Ontology based technique to identify designing pattern is proposed in [4]. Pattern test case templates (PTCT) are used for identification of structural test cases for these patterns. (PTCT) is described as reusable test cases. Such technique was discussed in [13]. GUI pattern based testing is also another direction of test patterns discussed in [2] [15]. A technique proposed in [2] suggests the use of GUI pattern to generate test case for recurrent situation. Another technique proposed on [14] uses GUI pattern for anti-pattern testing technique. [16] [15] authors describe the practical use of GUI pattern for mobile systems. Along with these patterns, defect patterns have also been used to support testing [8][9]. The concept of test pattern was proposed in [1]. However, its structural definition and practical use description is limited.

The activity of writing test first then code expressed as Test First (TF) is proposed in [20]. A cycle of test then develop property provides a short cycle of development [21]. Several researchers have studied TDD and compared it with traditional approach to enlighten these aspects [21] [28]. Such comparison was comprehensively discussed in [21] where the authors show on a scale of 24 qualitative and 16 quantitative features the strengths of TDD. They proposed that TDD have major support of reusability. Flexibility, effectiveness, risk reduction is highly improved. Moreover, the complexity and rework cost have been reducing while using the TDD. In TDD productivity and response to stake holder need to be a better response [14]. Another comparison was proposed in [20] where authors prove that there is no difference between the TLD and TDD. However, they have additionally stated that TDD has shown more biasness towards positive test cases.

Cloud based systems have a number of vulnerabilities. A taxonomy of potential threats on the basis of 4 layers; infrastructure layer, platform layer, application layer and administration was proposed in [23]. The authors describe 23 threats related to these categories. These threats represent misuse case of cloud [25]. In order to handle these threats several researchers have described patterns. A Fire Wall based pattern i.e. (Cloud Web Application Fire wall pattern (CWAF)) to handle SQL injection and other data vulnerabilities were proposed in [24]. The authors have proposed VM repository and cloud policy management point. Concept of countermeasure pattern and Threat patterns in cloud was proposed in [26].

Considering threats shared in [23], we provide a list of categories and sub categories and our analysis of positive and negative situations of use in Table I. This allows us to do processing such that we are able to identify sub category and super category. The identification of these risks allow us to provide test patterns. We provide test patterns for each

category. However, we include only one of them for the purpose of brevity since inclusion of test patterns for all of the categories is not possible. Test patterns based testing provides basis and convenience for TDD and TLD at the same time.

V. CONCLUSION

In this paper, we propose an approach of test patterns for testing. Test patterns provide template for testing based on collection of testing best practices and industrial experiences and can help in achieving high quality. We study previously identified cloud application threats and provide a mapping of categories, sub-categories, positive and negative use situations. This helps us identify test patterns which can be applied to individual testing situations where we require to know which patterns are applicable. This helps us to finally arrive at test cases. Our approach provides a homogenous ground for testing activity. It provides support for TDD and TLD.

As an outlook, we plan to link our proposal with specification level concepts such as use cases and misuse cases. We also plan to test our approach on bigger case studies and examples.

REFERENCES

[1] Wang Yi-chen, Wang Yi-kun (2011, June). The Research on Software Test Pattern. Future Computer Sciences and Application (ICFCSA), 2011 IEEE (pp.109 – 113).

[2] Moreira, R. M., Paiva, A. C., &Memon, A. (2013, November). A pattern-based approach for GUI modeling and testing. In Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on (pp. 288-297). IEEE.

[3] Coimbra Morgado, I., Paiva, A. C., & Faria, J. P. (2014, September). Automated Pattern-Based Testing of Mobile Applications. In Quality of Information and Communications Technology (QUATIC), 2014 9th International Conference on the (pp. 294-299). IEEE.

[4] Thongrak, M., &Vatanawood, W. (2014, July). Detection of design pattern in class diagram using ontology. In Computer Science and Engineering Conference (ICSEC), 2014 International (pp. 97-102). IEEE.

[5] Bozic, J., &Wotawa, F. (2014, March). Security testing based on attack patterns. In Software Testing, Verification and Validation Workshops (ICSTW), 2014 IEEE Seventh International Conference on (pp. 4-11). IEEE.

[6] Jürjens, J. (2002). UMLsec: Extending UML for secure systems development. In ≪ UML ≫ 2002—The Unified Modeling Language (pp. 412-425). Springer Berlin Heidelberg.

[7] Alvi, A. K., &Zulkernine, M. (2012, August). A Comparative Study of Software Security Pattern Classifications. In Availability, Reliability and Security (ARES), 2012 Seventh International Conference on (pp. 582-589). IEEE.

[8] Li, N., Li, Z., & Zhang, L. (2010, May). Mining frequent patterns from software defect repositories for black-box testing. In Intelligent systems and applications (ISA), 2010 2nd international workshop on (pp. 1-4). IEEE.s

[9] Li, M., Zhang, H., Wu, R., & Zhou, Z. H. (2012). Sample-based software defect prediction with active and semi-supervised learning. Automated Software Engineering, 19(2), 201-230.

[10] Sindre, G., &Opdahl, A. L. (2005). Eliciting security requirements with misuse cases. Requirements engineering, 10(1), 34-44

[11] Ikram, N., Siddiqui, S., & Khan, N. F. (2014, August). Security requirement elicitation techniques: The comparison of misuse cases and issue based information systems. In Empirical Requirements Engineering (EmpiRE), 2014 IEEE Fourth International Workshop on (pp. 36-43). IEEE.

[12] Yoshioka, N., Washizaki, H., & Maruyama, K. (2008). A survey on security patterns. Progress in informatics, 5(5), 35-47.

[13] Patterns: from system design to software testingY. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740-741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[14] Kaur, H., & Kaur, P. J. (2014, September). A GUI based unit testing technique for antipattern identification. In Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference- (pp. 779-782). IEEE.

[15] Coimbra Morgado, I., Paiva, A. C., & Faria, J. P. (2014, September). Automated Pattern-Based Testing of Mobile Applications. In Quality of Information and Communications Technology (QUATIC), 2014 9th International Conference on the (pp. 294-299). IEEE.

[16] Costa, P., Paiva, A. C., & Nabuco, M. (2014, September). Pattern based gui testing for mobile applications. In Quality of Information and Communications Technology (QUATIC), 2014 9th International Conference on the (pp. 66-74). IEEE.

[17] Correa, A. L., Werner, C. M., & Zaverucha, G. (2000). Object oriented design expertise reuse: An approach based on heuristics, design patterns and anti-patterns. In Software Reuse: Advances in Software Reusability (pp. 336-352). Springer Berlin Heidelberg.

[18] Bozic, J., & Wotawa, F. (2013, May). Xss pattern for attack modeling in testing. In Proceedings of the 8th International Workshop on Automation of Software Test (pp. 71-74). IEEE Press.

[19] Parveen, T., & Tilley, S. (2010, April). When to migrate software testing to the cloud. In Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on (pp. 424-427). IEEE

[20] Cauevic, A., Punnekkat, S., & Sundmark, D. (2012, September). Quality of testing in test driven development. In Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the (pp. 266-271). IEEE.

[21] .Bajaj, K., Patel, H., & Patel, J. (2015, October). Evolutionary software development using Test Driven approach. In Computing and Communication (IEMCON), 2015 International Conference and Workshop on (pp. 1-6). IEEE.

[22] .Kumar, S., & Bansal, S. (2013). Comparative study of test driven development with traditional techniques. Int J Soft Comput Eng (IJSCE),3(1), 2231-2307.

[23] Hashemi, S. M., &Ardakani, M. R. M. (2012). Taxonomy of the security aspects of cloud computing systems-a survey. Networks, 2, 1Virtualization.

[24] Fernandez, E. B., Yoshioka, N., &Washizaki, H. (2014). Patterns for cloud firewalls. AsianPLoP (pattern languages of programs), Tokyo.

[25] Fernandez, E. B., Monge, R. A. U. L., &Hashizume, K. E. I. K. O. (2013, October). Two patterns for cloud computing: Secure virtual machine image repository and cloud policy management point. In 20th conference on pattern languages of programs (PLoP 2013), Monticello, IL.

[26] Hashizume, K., Yoshioka, N., & Fernandez, E. B. (2011, October). Misuse patterns for cloud computing. In Proceedings of the 2nd Asian Conference on Pattern Languages of Programs (p. 12). ACM.

[27] Okubo, T., Wataguchi, Y., &Kanaya, N. (2014, August). Threat and countermeasure patterns for cloud computing. In Requirements Patterns (RePa), 2014 IEEE 4th International Workshop on (pp. 43-46). IEEE.

[28] Fucci, D., & Turhan, B. (2013, October). A replicated experiment on the effectiveness of test-first development. In 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (pp. 103-112). IEEE.