Indian Journal of Science and Technology, *Vol 8(21), DOI: 10.17485/ijst/2015/v8i21/69958, September 2015*

ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Analysis of Various Types of Bugs in the Object Oriented Java Script Language Coding

F. Fawzia Khan[1*] and R. Mallika[2]

[1]Department of Computer Science, Karpagam University, Coimbatore - 641021, Tamilnadu, India; fawziakhanphd@gmail.com
[2]Department of Computer Science, CBM College, Coimbatore - 641042, Tamilnadu, India

Abstract

Objective: The main goal of this work is to categorize and understand the various types of bugs present in the Object oriented java scripting language. By finding these faults one can find out the nature of faults which causes the run time failure of programs. **Methods:** There are various types of bugs present in the OOJS environment which needs to be categorized well to understand the nature of fault which has occurred in the OOJS language. By doing so, web pages can be prevented from the functioning failure and can cause the generation of more user flexible environment. Fault Localization is the approach to analyse and detect the place of faults present in the java scripting language. Transductive Support Vector Machine (TSVM) classification algorithm is introduced to categorize the faults into various sub types by classifying them based on types of bugs. This classification is done on the bug report data set which was created by using the fault localization approach. **Findings:** The experimental test conducted proves that the proposed approach named TSVM based categorization of faults can detect the faults efficiently which can be used further for error detection. The proposed approach improves in its performance in terms of improved accuracy detection by categorizing the faults correctly through which faults present in the programming language can be done efficiently. **Conclusion:** From the findings it can be concluded that the proposed approach improved in its performance in terms of improved accuracy.

Keywords: Bug Report, Fault Localization, Object Oriented Java Scripting Language

1. Introduction

Object Oriented Java Script is a scripting language which is used to provide a user interaction form in a dynamic manner to the web pages for the clients. The Object oriented java scripting language is used to provide the user interaction screen to the clients to navigate and do some dynamic operation over the network. The Object oriented java script language strictly differs from the java programming where the java is a real time programming language. Whereas, the Object oriented java script is a way to provide a logical communication to the web pages. Object oriented java script cannot be written directly as the separate coding. It can only be written inside HTML language which enables the users who are accessing the corresponding web pages to download and view the Object oriented java script content. By doing so, a more flexible environment can be created through which users can learn the information about the web page which are accessed by some set of users.

The Object oriented java scripting language can be learned more quickly and efficiently. However at the time of implementation it will be difficult to do the coding for creation of web pages which needs more focus in order to avoid the failures of web page creation. At the time of creation of web pages using the Object oriented java scripting languages there are many factors to be considered in order to avoid many security issues. One of the security issues which might possibly arise is corruption of the files which are stored in the user file system. This occurs due to the web page access behaviour of the innocent users. The end user may open some of the web pages unaware of Object oriented java script code existing in the corresponding HTML code. Generally Object oriented java script code will start to execute instantly at the time of triggering the page. At the time of triggering

* *Author for correspondence*

the web page, Object oriented java script can learn the information residing in the user file system. This needs to be prevented in order to avoid the user privacy violation from the Object oriented java scripting language.

Another security issue which may arise due to Object oriented java scripting language usage is that the web page developers may be unaware of the advanced features present in the Object oriented java scripting language. Due to this situation there may be possibility of the occurrence of the bugs at the time of program execution. These bugs cannot be analysed directly because of hidden logical file traces. These traces need to be identified locally in order to avoid the failure of the web pages execution.

The main contribution of this work is to locate and analyse the types of bugs which arise in the Object oriented java scripting language code creation and prevent from the web page failure during run time. The identified faults have been classified in accordance to the fault categories which can be used to identify the behaviour and nature of the fault. It is done by gathering the various bug reports from different web page developers. After gathering the bug reports, those reports are analysed to find out the traces and it is categorized as different types of bugs.

The organization of this work is given as follows: In section 1 detailed description about the introduction of the Object oriented java script language and the types of security issues which may arise in them is discussed. In section 2 previous researches has been discussed in a detailed manner to analyse the nature of faults which has been detected and analysed already. Section 3 provides detailed description about the proposed methodology in this work with the explanation of the overall flow of this research. In section 4 experimental tests which has been conducted before has been discussed in a detailed manner. Finally in section 5 results that are obtained are concluded in a manner of how it has improved in performance.

In[1], bug generations are found in the web applications in order to differentiate types of errors which can occur. It is done with the help of test case generation process and the explicit state model checking in which processing can be evaluated with the consideration of various functions and methodologies that are present in the system. Different types of test cases are generated in order to evaluate the performance of the software in which various bug fields can be enumerated. This is achieved by processing the various types of bugs and finding it over the web applications.

In[2], a way to prioritize the bugs in terms of crash report which has occurred in the different scenarios is discussed. This is done by identifying the types of bugs which happened in different location in terms of various methodologies. The prioritizations of bugs are identified with the knowledge of the crashes which may happen due to the corresponding bug. This is done by analysing and interpreting the various types of bugs that happened in different types of locations in terms of various methodologies. It is done by analysing and interpreting the failure which happened in the system.

In[3], distribution of bugs present in the particular type of software is identified. It is intended to analyse the various types of bugs and the distribution amount of those bugs in the software development environment. It is done by considering the various parameters which can lead software to a failure mode due to large amount of crashes present in the system. It is done by considering the percentage of crashes that occur in the system. This is achieved with the help of distribution function and the generative model. It is overcome by the consideration of various fields in which various types of bugs are distributed more.

In[4], the concurrent bugs are identified in order to predict the decision which needs to be taken on them to eliminate. It is done with the help of finding the similarity present among the various types of bugs which are present in the network. However it will be more difficult to identify the concurrent bugs, because of different formats of bugs. It is done by achieving the similarity retrieval over the various types of bugs. This is done by finding the synchronization intention present among the different bugs. To do so, invariant is identified which is present among the different types of networks.

In[5,14], the bug report can be created only by identifying the different types of bugs that are present in the system. It is done by considering the various types of bugs which is happening in the different types of system. In this research, bug reports are created by localizing the fault and create them in accordance to the various fields. Fault localization is generally achieved by using the software testing and designing phase. And also identified bugs are analysed and removed by using the approach called coincidental correctness. It is done by analysing and identifying the various types of fields.

In[6], multiple ways for constructing the software without error and constructing the good base software is discussed. In this work, various studies are conducted in terms of software availability and the software requirements that are elicited. Efficient software can be developed with the help of uniqueness of software characteristics. Good software is differed from other software in term of its security level. Here security level is defined in terms of the software failure which will occur at the time of software compilation. And also, in this work, security level is divided into two levels and concentrated individually at the time of software development.

In[7], ways for building secure and reliable software with the consideration of the software development is discussed. Reliable software development can be obtained with the consideration of the requirements which are gathered from the different phases in terms of previous history of similar type of software. This analysis is done by the author in the product which he has developed in his laboratory and the effectiveness is proved with the consideration of all types of security requirements. This research results in an effective way for constructing the software with more reliability without software failure.

In[8,13], a novel approach is introduced to gather the software requirements from the various phases which intend to produce good quality software. Good quality software can be obtained with the consideration of the complete software requirements. However it is more difficult to gather the software requirements which are more relevant to the software which is yet to be developed. In this work, personal driven approach is introduced which aims to satisfy the users with the requirements consideration, which was gathered from them. To achieve this user interaction interface is created which is named as the persona, and through which the user requirements are gathered and processed effectively.

In[9], an agile methodology is introduced to produce more secured software which can lead to a successful completion of software development. The agile methodology leads to a more effective implementation of the software development which is based on the incremental and iterative software development. This methodology will gather the requirements at each and every phase at the time of software development in order to improve the software. The software development may lead to a successive iteration of every cycle by gathering the requirements based on the software development stage.

In[10], a novel approach for gathering the requirements in terms of reuse methodology is discussed. A novel methodology to gather and process the requirements in terms of the software development will be useful only for the corresponding software which is going to be developed whereas in this work, software is developed with the consideration of reuse methodology. The requirements gathered for the software development will be maintained in the database and then it will be used further for other similar types of software. And also the characteristics of the current software also will be gathered and stored in database to maintain good configurable software.

2. Object Oriented Java Script Bug Analysis and Classification

Object oriented java script differs from the other coding languages in two ways[11]. Those are abstraction and encapsulation. The abstraction and encapsulation process is defined as; the object that is created can be used to extract the properties and methods of the external objects by inheriting them. Most Java Script libraries that you can obtain to make your java script coding easier uses Object oriented java script within the library itself in order to make it easy for people to perform the tasks that the library is designed to provide. Abstraction means that once you start using a library of objects designed to perform given tasks you no longer need to worry about exactly how to perform those tasks without the library. You simply call the appropriate methods for the appropriate objects and the library does the rest[12]. The code within the objects in the library is encapsulated so that the exact way in which they implement that functionality doesn't affect the way you call it. The library authors can rewrite code within the objects in their library and provided that they don't change the public interface you don't need to change any of your code to use their modified objects.

In this research different types of bugs which have occurred in the Object oriented java script programming language are attempted to be identified and analysed. Object oriented java script is a client side scripting language which may consist of the various types of the functions and modules to provide an advanced feature for the creation of the well-defined web pages. The bugs present in the Object oriented java script language may lead to a failure of web page creation and will reduce the user's satisfaction level. Thus analysing of possible faults which may arise in the Object oriented java script language and classifying them according to the fault categories are more important in the case of real scenario. In this work this problem is resolved by introducing the methodology called the fault localization methodology which aims to analyse the number of faults occurred in the Object oriented java scripting language with the detailed information of the fault and fault localization. Here, TSVM classification methodology is introduced which aims to classify the faults based on its categorization. TSVM is the classification methodology which can able to classify the data points that are partially labelled. TSVMs are basically iterative algorithms that gradually search the optimal separating hyper plane in the feature space with a transductive process that incorporates unlabeled samples in the training phase. This procedure improves the generalization capability of the classifier. Various types of fault categories are analysed here. The types of fault categories that are assumed are:

2.1 Undefined/Null Variable Usage

Trying to access an Object oriented java script variable which has been not declared or declared without assignment of values may lead to a bug. That is accessing objects or methods which have been defined without values. For example, trying to access a variable x using the property bar, x.bar which is not been declared in the Object oriented java script code.

2.2 Undefined Method

As like previous undefined method bug will arise at the time of accessing of methods which has not been declared before in the Object oriented java script code. For example trying to calling a method food() which has not been declared in the Object oriented java script code.

2.3 Incorrect Method Parameter

This type of bug will arise at the time of passing wrong values in to the methods which is defined. The parameters that are to be invoked will be defined at the time of function declaration. If it is done wrongly at the time of function calling by sending wrong values, then the bug will be created. For example, passing a string value to the setDate() function instead of sending the integer value which will lead to a failure of software execution.

2.4 Incorrect Return Value

If the values returned are generated wrongly due to some minor mistakes existing in the logical programming, this type of bug will be raised.

2.5 Syntax-based Fault

If the programming is done without following the syntax rules defined in the Object oriented java scripting language bin files, then this type of errors will occur. For example; instead of double quotation we can use single quotation to define a word.

2.6 Range based Fault

This type of fault will occur when the passing parameters values for the particular attributes resides in arrange of values. That is varying of data values based on the parameter range exists among them.

2.7 Incorrect Object Support

Defining or extracting the objects which are not relevant to the concept of the methods or functions. The methods and properties need to be defined properly.

2.8 Other

There are some errors occurred other than the errors which are defined above. For example, the naming conflict present among the function and attributes that are defined.

The above fault categories are concentrated in this research for better detection of the bugs present in the Object oriented java scripting language.

The bug detection and classification algorithm is works as follows:

- Detect the place of errors existing in the Object oriented java scripting language code.
- Localize the errors and then analysing them for detecting the reason for the occurrence of particular fault.
- And then find the source of fault by calling the function, in which the error has occurred.
- Then classifying the errors by using the TSVM machine learning algorithm.

2.8.1 Error Localization

First the errors present in the Object oriented java scripting needs to be analysed in order to find the nature of fault which can cause the program execution failure. This is done in two steps:

- Error collection.
- Error analysis.

These two steps are followed to find the exact place where the error has occurred. In the error collection phase, the errors in the Object oriented java scripting coding will be analysed and identified in an iterative manner by executing them with the minor changes. After collection of the error evidences, those errors will be analysed to know what type of errors has occurred and the nature of the evidences. These two phases are discussed in detail in the following section.

2.8.1.1 Error Collection

In the error collection phase, the Object oriented java scripting code will be analysed and searched to identify the presence of error in the corresponding code. This is done by analysing the corresponding error present in the Object oriented java script. The error may be generated at the time of coding creation which may lead to software failure. Thus every line of the code needs to be analysed at the time of execution of coding. The errors in the Object oriented java script coding are collected based on:

- The error line is present in which function;
- The function to which the corresponding line number belongs to;
- Variable and functions in the error lines;
- The previous values of the corresponding variable, in the previous execution.

After collecting these errors which are present in the Object oriented java scripting language, the nature of errors will be identified and the reason behind those errors occurrences will be identified. By doing so, the synchronous errors present in the Object oriented java scripting coding will also be identified.

2.8.1.2 Error Analysis

After collection of errors and its location in the Object oriented java script code, the analysis of the errors will be done to identify the nature and behaviour of the errors. By doing so, the document Object model access which is responsible for the corresponding error in the Object oriented java scripting will be identified. That is, root and source of the bug will be identified and then it will be analysed for the future prevention from bugs. This is done by analysing the exact part of the coding which is responsible for the failure of the web page creation, by segmentation of the Object oriented java script error code into partitions.

After segmentation of errors in to the partitions, the relevant features will be identified by looking for the DOM access control. Initially, errors will be analysed in the document by using an error marker and will take an error as an event. The error will always be observed in the relevant sequence, since the program will be halted once the error occurs. That is, in this phase, the sequence of events which are reason for the corresponding bug will be identified in order to analyse and remove them for providing the convenient environment for the users. The corresponding errors will be analysed for the removal of such noise, thus the efficient and bug avoided Object oriented java scripting can be implemented. After this process, the identified bugs from the various modules and functions will be categorized and learned for the future use which can be used to avoid the same type of failure at the time of project execution.

2.8.2 Failure Categorization

After analysis of different types of bugs present in the Object oriented java scripting language, the identified bugs will be categorized in accordance to the types of faults which was discussed in the previous section. This classification is done by using the classification algorithm called Transductive Support Vector Machine which aims to classify the bugs that are identified based on their type and nature. This TSVM algorithm is used to avoid the convergence problem which may occur due to the large volume of data.

TSVM is the iterative algorithm which aims to cover the unlabelled data present in search space with the help of transductive function by separating the hyper plane optimally. In the training phase of the TSVM, unlabelled data will be considered for the classification. That is the bugs without label will be considered in the training phase for the better classification process. By using the TSVM methodology, better generalization capability of the classifier can be achieved. Also, this methodology aims to move the hyper plane gradually based on reaching a finer place. This methodology aims to achieve a finer place by achieving the generalization capability of the classifier. A finer hyper place position can be achieved by iteratively doing this mechanism. In this approach, better classification can be achieved by migration of the data into the class labels which pretend to avoid the misclassification of the data by avoiding the discriminate functions. This mechanism also aims to provide more accurate results than the existing methodology. The convergence of the categorization depends on the similarity present among the different class labels.

The designing of TSVM for the fault categorization is done in two ways and it addresses two issues. Those are:

- Select the bugs with the expected accurate labelling.
- Choose the informative knowledge source about the Object oriented java script bug.

TSVM selects the margin of the classification of bug samples by accurately identifying the upper and lower side of the samples by analysing and checking the following conditions. Those are, if $P \geq 1$ (ie., falls below or above the hyper plane region will be identified), then the transductive samples of bugs closest to the margin bounds will be assigned as class labels as +1 or -1 respectively. Else labelling will be done any way without any consideration.

A dynamic adjustment is necessary, taking into account that the position of the hyper plane keeps changing at the iteration. Typically, the most confident unlabelled patterns, together with their predicted labels, are added to the current training set. The classifier is retrained and the process is repeated. It is to be noted that the classifier uses its own prediction to teach itself. It is natural to imagine that a classification error can reinforce itself. Therefore, it is important to take a caution in the selection of transductive samples because wrong labelling may substantially degrade the performance of the classifier.

Due to the fact that support vectors contain the richest information among the informative samples (i.e., the ones in the margin band), the unlabelled patterns closest to the margin bounds have the highest probability to be correctly classified. Therefore, in the proposed approach, we design a selection procedure (i.e., filtering process) to increase the acceptability of the samples with the expected correct labelling. In other words, an unlabelled sample should be considered as transductive sample if the TSVM ensemble assigns the same label to it. We can expect this sample bearing the information with an expected accurate class label.

2.9 Algorithm

2.9.1 Input

Labelled points: $S = [(\mathbf{x}_j, y_j)]$, $j = 1, 2, \ldots, l$ and unlabelled points: $V = [(\mathbf{x}_j)]$, $j = l + 1, \ldots, n$.

2.9.2 Output

TSVM classifier with original training set and the transductive set.

2.9.3 Begin

```
1. Initialize the working set $W^{(0)} = S$, previous transductive
   set $A_t^{(0)} = \phi$ and specify C and $C^*$.
2. Train SVM classifier with the working set $W^{(0)}$.
3. Obtain the label vector of the unlabeled set V.
   For i = 1 to T // T is the number of iterations.
4. Select N+ positive transductive samples from the
   upper side of the margin and N– negative transductive
   samples from the lower side respectively.
5. Select positive candidate set B+ containing N+
   positive transductive samples and negative candidate
   set B – containing N– negative transductive samples
   respectively.
6. $B_t^{(i)} = B^+ \cup B^-$
7. Update the training set:
    If $A_t^{(i-1)} = \phi$ $W(i) = W^{(i-1)} \cup B_t^{(i)}$
        $D_t^{(i)} = B_t^{(i)}$
    Else
        $D_t^{(i)} = A(i-1)t \cap B(i)t$
        $W^{(i)} = (W^{(i-1)} - D_t^{(i-1)}) \cup D^{(i)}t$
End if
8. $A_t^{(i)} = B_t^{(i)}$
9. Train TSVM classifier with the updated training set W
   (i).
10. Obtain the label vector of the unlabelled set V.
End for,
End,
```

The above classification algorithm is used to result the categorization of the failures due to bugs based on the fault types. Thus with the knowledge of this algorithm one can analyse the detection of bug that are present in the Object oriented java scripting language coding easily.

Finally, experimental tests has been conducted to analyse the types of bugs have been present in the Object oriented java scripting language is predicted correctly or not.

3. Experimental Results

The experimental test has been conducted to compare the effectiveness of the algorithm in order prove the improvement in the proposed research than the existing research named AUTOFLOX which aims to localize the faults that are occurring in the Java Script language automatically. The comparison is done based on the performance metrics called the precision and the recall measures.

In this work, Object oriented java script based bug reports have been collected from the various web page developer which consists of details about the various types of bugs which may occur in the different situations. By analysing the bug report, the Object oriented java script bug report prediction is done and the types of faults are categorized. The types of bug which are present in the Object oriented java script coding has been classified in accordance to the nature of the bugs.

The comparison based on these performance metrics are explained in the following sections:

3.1 True Positive Rate (TP)

It is the amount of correct faults that are classified to correct class.

$$TP=\frac{d}{c+d}$$

3.2 False Positive Rate (FP)

It is the amount of negative faults that are classified into correct class to which it does not belong to:

$$FP=\frac{b}{a+d}$$

3.3 False Negative Rate (FN)

It is the amount of negative faults that are classified to wrong class.

$$FN=\frac{c}{c+d}$$

Where,

- a is the number of correct predictions that an instance is negative.
- b is the number of incorrect predictions that an instance is positive.
- c is the number of incorrect of predictions that an instance negative.
- d is the number of correct predictions that an instance is positive.

These are all the measures which are used to calculate the accuracy values.

3.3.1 Precision

Precision value is used to indicate the successful prediction of bug report creation based on the true positive and the false positive information. The equation to calculate the precision value is defined as follows:

The table that lists the actual precision values that are obtained while processing the java scripting bug corpus is given in Table 1.

$$Precision=\frac{\text{True positive}}{\text{True positive+False positive}}$$

Table 1. Precision values

Number of bugs	PRECISION	
	AUTOFLOX	TSVM
10	0.96	1
20	0.92	0.98
30	0.79	0.87
40	0.75	0.81

The comparison of precision value between proposed system and the existing system are shown in the following graph:

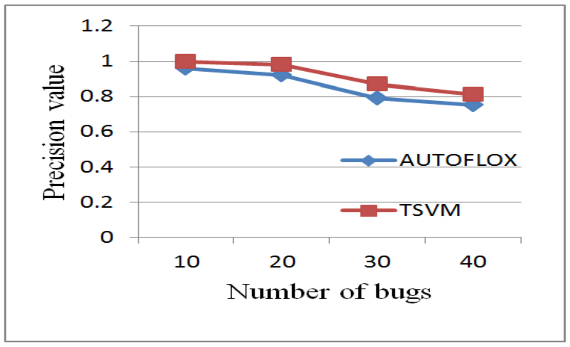

Figure 1. Precision comparison.

From the Figure 1, it is proved that the precision obtained in the existing work is lower than the proposed methodology. In x axis, the number of bugs that are analyzed are taken and in y axis precision value is considered.

3.3.2 Recall

Recall value is calculated based on the successful prediction at true positive prediction and false negative. Recall value is calculated by using the following equation:

The table that lists the actual recall values that are obtained while processing the java scripting bug corpus is given in Table 2.

$$\text{Recall}=\frac{\text{True positive}}{\text{True positive+False negative}}$$

Table 2. Recall values

Number of bugs	RECALL AUTOFLOX	RECALL TSVM
10	0.8	1
20	0.76	0.98
30	0.72	0.87
40	0.61	0.81

The comparison graph for recall value is depicted in the Figure 2.

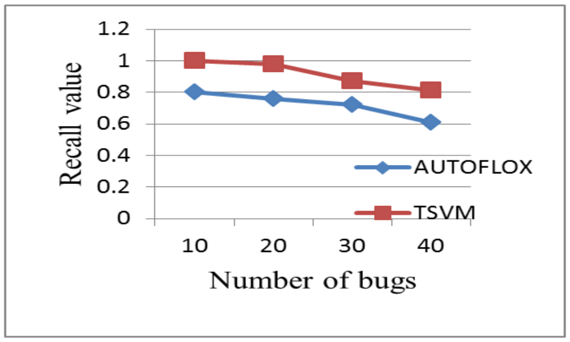

Figure 2. Recall comparison.

Form the Figure 2 it is proved that the recall value obtained in the existing work is lower than the proposed methodology. In x axis, the number of bugs that are analyzed are taken and in y axis recall value is considered.

From this analysis it can be proved that the proposed methodology can identify the faults accurately than the existing approaches from which it has been proved that the proposed approach provides 70% improvement than the AUTOFLOX approach.

4. Conclusion and Future Work

Object oriented java script bug handling is the most complex process in the real world web page development environment. The small errors which arise during development of the client side programming may lead to the entire corruption of the execution of the coding. Finding of bugs in the Object oriented java script coding will also be more complex process due to the small logical errors which cannot be identified. In this work, fault localization and the fault categorization methodology is introduced which aims to predict and classify the bugs that are present in the Object oriented java scripting code based on the nature and behaviour of the bugs. In our work, six types of bugs have been considered for the effective categorization of the bugs that are present in the document. The finding of this work demonstrates that the proposed methodology can lead to an efficient prediction of the bugs and their classification which provides a convenient way for the web page developers to avoid the bugs during run time.

In the future scenario, automatic classification of bugs occurring can be introduced through which the time complexity can be reduced considerably. The various prospects of the java script language need to be concerned more for supporting the higher technology software.

5. References

1. Artzi S, Kie_zun A, Dolby J, Tip F, Dig D, Paradkar A, Ernst MD. Finding bugs in web applications using dynamic test generation and explicit-state model checking. IEEE Trans Software Eng. 2010 July-Aug; 36(4):474–94.
2. Kim D, Wang X, Kim S, Zeller A, Cheung SC, Park S. Which crashes should I fix first?: Predicting top crashes at an early stage to prioritize debugging efforts. IEEE Trans Software Eng. 2011 May-June; 37(3):430–47.
3. Concas G, Marchesi M, Murgia A, Tonelli R, Turnu I. On the distribution of bugs in the eclipse system. IEEE Trans Software Eng. 2011 Nov-Dec; 37(6):872–7.
4. Lu S, Park S, Zhou Y. Detecting concurrency bugs from the perspectives of synchronization intentions. IEEE Trans Paralell Distr Syst. 2012 June; 23(6):1060–72.
5. Zhang Z, Chan WK, Tse TH. Fault localization based only on failed runs. IEEE Computer Society. 2012; 45(6):64–71.
6. McGraw Gary. Building secure software: Better than protecting bad software. IEEE Software. 2012 Dec 16; 19(6):57–8.
7. Fichtinger B, Paulisch F, Panholzer P. Driving secure software development experiences in a diverse product environment. IEEE Computer and Reliability Societies. 2012 March-April; 10(2):97–101.
8. Cleland-Huang J. Meet elaine: A persona driven approach to exploring architecturally significant requirements. IEEE Software, IEEE Computer Society. 2013; 18–21.
9. ben Othmane L, Angin P, Weffers H, Bhargava B. Extending the agile development approach to develop acceptably secure software. Journal of IEEE Transactions on Dependable and Secure Computing. 2014; 11(6):497–509.
10. Hermoye LA, van Lamsweerde A, Perry DE. A reuse-based approach to security requirements engineering. Proceedings of 9th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'03); 2003.

11. Artzi S, Dolby J, Jensen S, Moller A, Tip F. A framework for automated testing of java script web applications. ACM in International Conference on Software Engineering (ICSE); 2011. p. 571–80.
12. Ocariza F, Pattabiraman K, Zorn B. Java Script errors in the wild: An empirical study. Proceedings of International Symposium on Software Reliability Engineering (ISSRE). IEEE Computer Society; Hiroshima. 2011 Nov 29-Dec 2. p. 100–9.
13. Groeneveld F, Mesbah A, van Deursen A. Automatic invariant detection in dynamic web applications. Delft University of Technology. Tech Rep; 2010.
14. Zheng Y, Bao T, Zhang X. Statically locating web application bugs caused by asynchronous calls. International Conference on the World-Wide Web (WWW), ACM; 2011. p. 805–14.