ISSN 2347 - 3983
Volume 8. No. 10, October 2020
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter1048102020.pdf
https://doi.org/10.30534/ijeter/2020/1048102020

Generating Test Cases from Mobile Application using Depth First Search and Backtracking Techniques

Tay Wei Han[1], Rosziati Ibrahim[2], Samah W.G. AbuSalim[3], Jahari Abdul Wahab[4]
[1]Universiti Tun Hussein Onn Malaysia, Malaysia, ai170223@siswa.uthm.edu.my
[2]Universiti Tun Hussein Onn Malaysia, Malaysia, rosziati@uthm.edu.my
[3]Universiti Tun Hussein Onn Malaysia, Malaysia, samahwasalim@gmail.com
[4]SENA Traffic Systems Sdn. Bhd., Malaysia, jahari@senatraffic.com.my

ABSTRACT

Software testing is a part of the process in software development life cycle (SDLC). Nowadays, the size of software developed grow larger and bigger. The cost of testing with manual method which is not efficient become significant. A good algorithm or method which is less time consuming and lower in cost is important to solve this problem. However, although there are different kinds of algorithm available for us to apply, which algorithm is better for generating test cases is still not confirmed. This research is carried out in order to find out and compare two algorithms which are Depth First Search and Backtracking. The time taken for both algorithms to generate test cases or test pathways are recorded and compared. Three applications were used as three cases to gather more results in different size of application. The results show that the Backtracking algorithm can achieve better and faster runtime results due to the algorithm require lesser loops than the Depth-First-Search algorithm while the number of loops and test pathways are fixed.

Key words: Software Testing, Test Cases, Depth First Search (DFS), Backtracking Algorithm

1. INTRODUCTION

With the development of software engineering and the spread of its software in our daily life, strict requirements arose regarding the reliability, maintenance, and security of these programs [1]. When developing software, it is very important to ensure that the program is free from defects. However, not all programs can be 100% free from defects [2]. The industry had to respond to these new requirements by developing methods of testing these programs, which means increasing the technical expertise of test engineers in addition to enhancing the importance and necessity of the test for programmers. Software testing is the main method used by many companies and industries to evaluate and improve the quality of the software being developed. The software testing process is carried out in three stages (generation of test cases, test execution of test cases and evaluation of test cases) to avoid the consumption of time as well as using many resources [3]. Hence, many testers use automation of software testing in order to save cost and time but can give results more accurate compared with software testing that have been done manually. The task of test case generation can be converted into an optimal problem by heuristic search techniques as Genetic Algorithm [4], Depth-first search (DFS) [5] [6], Hill Climbing methods [7] and A* Algorithm [8]. This research focus in implementing two algorithms namely Depth First Search (DFS) algorithm and Backtracking algorithm to generate test cases. The Depth First Search (DFS) is an algorithm, which used for searches, that is based on tree graph and uses the idea of backtracking. It explores all the nodes by going forward if possible or uses backtracking. It will search though all branches before backtracking [9]. It is a simple algorithm that is quite easy to understand and may give efficient results. For generating test cases, Depth-First Search is assumed that it may generate all available options within the program. Backtracking algorithm starts from the end of the tree. As each end of the tree will result in one unique test case path; by starting from each end nodes, it will reduce the number of loops, and reduce the deciding time. Backtracking may go through all possible nodes and make sure there is no any possible test cases paths left out [10].

This research aims to know which of Depth First Search technique or Backtracking technique is more suitable for generating test cases. A comparison will be made between both algorithms based on time consumed and complexity.

2. ALGORITHMS FOR TEST CASES GENERATION

In this section, Depth-First Search algorithm and Backtracking algorithm used to generate test cases automatically are presented.

2.1 Depth-First Search

Depth first Search (DFS) or Depth first traversal is a recursive algorithm for searching all the vertices of a graph or tree data structure. DFS is an efficient algorithm for exploring the tree, achieving the optimal cost of twice the sum of the lengths of edges in the tree [11]. Kirupa [12] stated that depth first search works by taking a node, checking its one side of branch, expanding the first node it finds among the branch, checking if that expanded node is our destination, and if not, continue exploring more nodes [9]. Depth first search is one kind of algorithm. Simply defined, this algorithm will go as deep as possible for one side of branch until it reaches the end, only then it will backtrack and go for another one side of the branch again, repeatedly. As this algorithm ensures that each node in the tree will be included, it is suitable to be used as a tool to generate test cases. However, the node that has been visited have to be recorded in the system to avoid duplicated test cases from occurring [13]. For example, in Figure 1, the first node "A", will be continued to "C" then "D", which is one side (left side) of the tree. Since "D" is the last node in that branch, the searching process will backtrack to "C" to search for another branch that have a node that is not visited yet. As "G" is not visited yet it will go to "G". Then, it will backtrack to "C" again, as there is no more node or branch that under "C" that have not been visited. The process will backtrack again, to "A" then "E", and "B". The flow will continue until all nodes have been visited which is "J" being the end as shown in Figure 1.

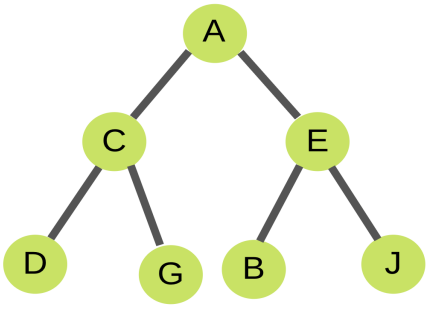

Figure 1: Depth-First Traversal

2.2 Backtracking Algorithm

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree). Backtracking Algorithm is usually applied to avoid or reduce the possibility of fail pathways in a maze or a puzzle [10]. Other than backtracking while fail in pathway, the backtracking technique may also be applied in the way to start its path from the end to find all possible path way to reach end from start by backtracking from end to start without leaving out any possible pathway [9].

3. RELATED WORK

This section explains on other related research reports that used different techniques to generate test cases. Although different techniques have been used, the aim of these research reports are similar which is to ease the way to generate test cases.

Hamimoune [14] applied four different mutation testing techniques. The techniques are random sampling, all operators, method-level operators and class-level operators. The results of [14] show that using all operators sampling was the best mutation testing technique. However, different mutation testing technique may affect in efficient and other factors such as all operators sampling only suitable for small and middle-sized applications. In larger application all operators sampling may consume a lot of time.

Srisura and Lawanna [15] shows a technique to improve regression testing by selecting suitable false test. Regression testing is known as a testing process that consume a lot of time. This may be more significant when using to test large application. This problem causing inefficient and high cost due to it is execute the test repeatedly. In this study, a false test case selection is proposed that select which part of false test which is really necessary to re-execute. Since part of the false test cases is excluded when re-execute the testing, the case size of testing will become smaller. So, the testing consumes shorter time and less cost.

Vivekanandan et al. [16] presented a new approach for test path generation automatically by using Clonal selection Algorithm and compare the results with Genetic Algorithm based on execution time. The results show that their approach reduces the effort and time of the route generation. Whereas, Clonal selection algorithm generate 85% of basis path with lesser time than genetic algorithm.

Banias [17] apply quadratic dynamic programming algorithm in a software testing domain, it makes the test case selection decision more specific. This may find potential defects in a shorter time, as all software testing work's time are limited. The method he uses calculate the priority in concern of time and cost.

Mishra et al. [18] shows a random test case generation by using genetic algorithm. It also studied how different types of hybridized genetic algorithm that are helped in software testing field by generate test cases randomly and automatically. Genetic algorithm here can be used with neural networks and fuzzy systems for performing different types of testing to improve the performance.

Arwan and Rusdianto [6] used two approaches to find on an independent path directly from a pseudocode. They use graph theory and Depth First Search algorithm for finding an independent path. The results show that the device accuracy check was able to find the right independent path.

Salihu et al. [19] implemented tool called AMOGA in order to test the mobile applications. The experimental results

show that AMOGA works to improve the coverage of mobile applications by creating a comprehensive model that does this and can be an alternative to the model-based testing of mobile applications. The technique has proved its efficacy by achieving a high degree of code coverage and mutation score for various applications.

Abusalim et al. [20] aim to employ an appropriate mutant reduction technique by comparing and analyzing between two approaches namely selective and non-selective mutants. The experimental results showed that selective mutant operators can drastically reduce the execution time and make mutation testing more effective since the execution time for selective mutant is 8.5 seconds while non-selective mutant takes 27.25 seconds in execution time.

Ibrahim et al. [21] proposed a DART approach to detecting and refactoring code smells from the Android application's source codes to minimize redundancy in test case generation. The results show that the number of test cases produced has been reduced and branch coverage was increased up to 5.0%.

Table 1 summarize the literature review in test cases generation techniques.

Table 1: The Comparison of Different Techniques

Authors & year	Technique	Definition	Contribution
Hamimoune . (2016) [14]	Mutation Technique	The best operator of Mutation technique is found by doing statistic comparison.	The concept of turning test object into nodes is referred.
Srisura and Lawanna. (2016) [15]	Regression Technique	Propose a new technique to improve current regression testing which minimize the case size.	The objective of this project is similar, to find or create algorithm that suits generating test cases.
Banias. (2017) [17]	Generic Algorithm	Ways to generate random test case and flow of genetic algorithm is proposed.	The format of test case generated is referred.
Mishra et al. (2018) [18]	Dynamic Algorithm	A calculation of the test cases priority is proposed	The importance of time taken that affect the testing
		and coding for checking test case priority is proposed.	process is shown.
Salihu et al. (2019) [19]	Model-Based testing	AMOGA isused to test the mobile applications	AMOGA improve the code coverage of mobile app
Abusalim et al. (2020) [20]	Mutation Technique	compare and analyze between selective and non-selective mutants for reduction technique	Selective mutant operators can drastically reduce the execution time

Based on Table 1, there are many researches proposed by several people on the field of software testing. Many researchers tried to reduce the time of generating test cases.

4. UML SPECIFICATION

UML diagram is used to illustrate the flow of project. The process of this research will be shown in figure within this section. All diagrams and figures follow the flow of process in generating test cases using specific algorithm. Firstly, the element of chosen application will be input manually by user, the type of element such as button, textbox will be defined by user. After all element of that application are entered as input by user in the correct flow, the user may choose to generate test cases by using either algorithm. The test cases will be generated using the chosen specific algorithm. In the end, the time taken of the result to be generated will be shown in notification. These test cases include the common function such as sign in, searching and edit profile information. The first diagram shown below is the use case diagram which takes place in designing system with actors, use cases and relations.

4.1 Use Case Diagram

Use case diagram graphically shows an overview of a system's flow and processes. The actor in this use case is a user. The user needs to insert the elements in correct flow into system, which in the interface created by Eclipse IDE environment. The use case includes insert elements, check dependencies of the elements, choose algorithm, and generating test cases. The figure that shows the use case diagram for generating the test cases using specific algorithm is shown in Figure 2.

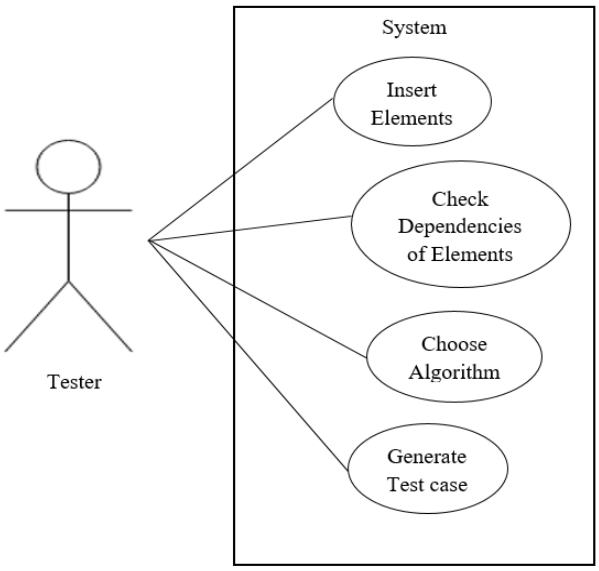

Figure 2: Use-Case Diagram

4.2 Class Diagram

Class diagram helps in constructing executable coding. It shows the system's structure by describing the system classes, attributes and also the operations. The figure of the class diagram for this case study is shown as on Figure 3.

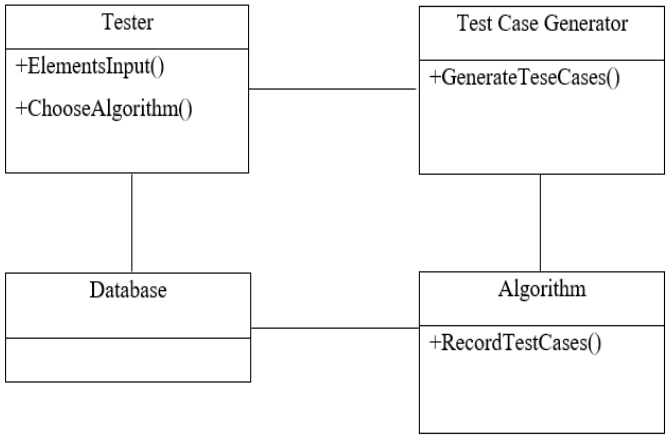

Figure 3: Class Diagram

5. RESEARCH METHODOLOGY

The steps for this research start with literature review on different methods and techniques that used for test cases generation., implement algorithms of DFS and Backtracking to generate test cases, and compare the time taken of both algorithms to complete generating test cases. Figure 4 shows all the steps for developing test cases generation tool. The following sub-sections discuss in details.

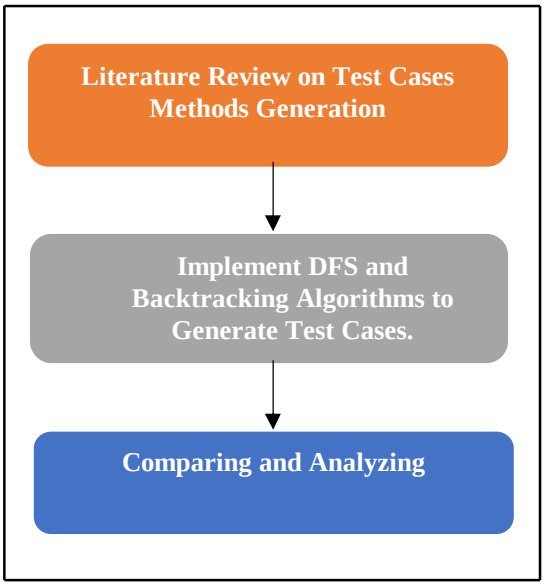

Figure 4: Research Methodology

5.1 Literature Review on Test Cases Methods Generation

This research follows the steps as in Figure 4. The literature review is conducted based on recommendation by Brereton [22]. Some of the related papers are reviewed as summarized.

5.2 Implement DFS and Backtracking Algorithms to Generate Test Cases.

Depth-First Search algorithm and Backtracking algorithm are used in developing the test cases generation tools. The functions and test data will be input by tester and the test cases generation tools will rearrange available data into sets of test cases. The list of test cases generated will act as the output by this tool. There will be knowledge of pointer and for loops applied in this generation tool. The initial possible tree is a required input by tester, only the tools may function to change them to numbers of test cases that includes every of each element. In this research, the Eclipse environment will be used to develop the test case generation tools. Eclipse is an integrated development environment (IDE) that is widely used for Java programing language [23].

5.3 Comparing and Analyzing

Depth-First-Search and Backtracking algorithm are both completed with the same numbers of test pathways. Implementation of Depth-First-Search and Backtracking algorithm on the test pathway generating tools are compared using the complexity and runtime with the same input number of nodes, ways and tree structure. Three case studies with different number of nodes are used in this research which are Skype, Google Map and Food Panda. Table 2 shows brief description for those case studies and number of nodes for each one.

Table 2: The Case Studies

Case Study	Description	Number of nodes
Skype	app to speak and send text or voice messages, communicate with friends and speak with them by voice or voice and image.	14
Google Map	providing real-time information on traffic density and public transport using GPS and exploring local neighborhoods by learning about places to eat and drink and places you can visit wherever you are in the world.	50
Food Panda	collect and delivers the food order within 30min after the order was placed by app.	61

6. RESULTS AND DISCUSSION

Table 3 shows the runtime in Backtracking (BT) algorithm and Depth-First-Search (DFS) algorithm. Backtracking algorithm is faster than Depth-First-Search algorithm for Case 2 and 3.

Table 3: Comparisons for The Results

Application	Number of Nodes	Number of Test Path Ways Generated	Time Taken in BT	Time Taken in DFS	Different in *ms*
Skype (Case1)	14	10	1ms	1ms	0ms
Google Map (Case2)	50	38	2ms	4ms	2ms
Food Panda (Case3)	61	48	3ms	5ms	2ms

The difference in time taken is calculated by using the formula:

$$(|T_b - T_d|) \quad (1)$$

where, T_b is the time taken by Backtracking algorithm and T_d is the time taken by Depth-First-Search algorithm.

Overall, Backtracking algorithm is faster than Depth-First-Search algorithm in most cases. In Case 1, the number of nodes and test pathways generated could be too little to see the difference in the time taken to do so. However, Case 2 and Case 3 show the same difference, where the Backtracking algorithm is faster than the Depth-First-Search algorithm by 2ms. This is because the Depth-First-Search algorithm has a greater Big-O value compared to the Backtracking algorithm. Depth-First-Search algorithm requires loops for every each of the nodes while the Backtracking only loops for the number of end nodes. More loops consume more time with limited resources.

Table 4 shows the results obtained after running the two algorithms are different due to the difference in complexity. For algorithms of DFS and Backtracking go through all nodes to generate all test pathways from these nodes. For Backtracking Algorithm, it loops all end nodes and backtrack its stack from the upper level; the time complexity of backtracking in this case is O(N) where N represent the number of end nodes. However, DFS loops for all vertices and edges where the time complexity is equal to O(V+E); V is number of vertices and E is number of all edges. Based on the time complexity, we can conclude that the Depth First Search algorithm will take more time than the Backtracking algorithm as the number of end nodes is always less than the number of vertices, N<V. Thus O(N) will always be smaller than O(V+E).

Table 4: The Comparison for Complexity of the Algorithms

Algorithm	Time complexity
Backtracking	O(N)
Depth First Search	O(V+E)

7. CONCLUSION

The demand for software products is increasing, competition in the software market is becoming very strong, and software quality is becoming more and more important, which plays a vital role in the performance of the entire software system [24]. This research aims to implement two algorithms namely the Depth First Search (DFS) algorithm and the Backtracking algorithm to generate test cases, and compare the time consumed for each two algorithms in generating test pathways using Eclipse IDE. Both Depth First Search (DFS) algorithm and Backtracking algorithm work well in generating test cases or test pathways. The reason that causes a difference in the time taken, is the number of loops required to go through all nodes and generate an output of test cases. As the concept of Depth First Search algorithm requires a loop in every node to search all its branches, the number of loops required is equal to the number of nodes. In comparison to the Backtracking algorithm which loops for every end node and backtrack to the beginning node, the

number of loops is only equal to the number of end nodes. So, the DFS algorithm is slightly slower compared to the backtracking algorithm and the difference will become bigger if the application being tested is larger, which will surely have more access points or nodes. However, DFS algorithm is somehow more reliable when the application has cycle between loops. This is because DFS will only visit each node once and mark its track until the last node, hence no nodes will have a repeated visit. There is no similar constraint for the Backtracking algorithm that may cause infinite loops when nodes are given with cycle branches.

A more complex situation can be explored when comparing algorithms other than only increasing the size of application and number of nodes. For example, giving nodes with cycle branches. More algorithms such as Breath-First-Search can also be included in future studies.

ACKNOWLEDGEMENT

This project is funded by the Ministry of Education Malaysia under the Malaysian Technical University Network (MTUN) grant scheme Vote K234 and SENA Traffic Systems Sdn. Bhd.

REFERENCES

1. G. Kaur, P. Singh. **Test Case Generation Using UML Diagram.** *International Journal of Emerging Technologies in Engineering Research (IJETER)* Volume 1, Issue 2, July (2015).
2. R. B. Jadhav, S. D. Joshi, U. G. Thorat, A. S. Joshi. **Software Defect Prediction Utilizing Deterministic and Probabilistic Approach for Optimizing Performance through Defect Association Learning**. *International Journal of Emerging Trends in Engineering Research, Volume 8. No. 6, June 2020.*
3. M., I. Septian, R. S. Alianto, Daniel, & F. L. Gaol. **Automated Test Case Generation from UML Activity Diagram and Sequence Diagram using Depth First Search Algorithm**. *Procedia Computer Science, 116, 629–637, 2017.*
4. R. Khan, M. Amjad, & A. K. Srivastava. **Optimization of Automatic Generated Test Cases for Path Testing Using Genetic Algorithm**. *2016 Second International Conference on Computational Intelligence & Communication Technology (CICT), Ghaziabad, 2016, pp. 32-36, doi: 10.1109/CICT.2016.16.*
5. N. Rathee, R.S. Chhillar. **A Survey on Test Case Generation Techniques Using UML Diagrams**. *Journal of Software vol. 12, no. 8, pp. 页码, 2017.*
6. A. Arwan, D.S. Rusdianto. **Automation of Independent Path Searching using Depth First Search.** *Journal of Information Technology and Computer Science, Volume 3, Number 1, 2018, pp. 104-112.*
7. F. C. M. Souza, M. Papadakis, Y. Le Traon, & M. E. Delamaro**. Strong mutation-based test data generation using hill climbing.** *Proceedings of the 9th International Workshop on Search-Based Software Testing - SBST '16.*
8. D.B. Mishra, R. Mishra, K.N. Das, A.A. Acharya. **Test Case Generation and Optimization for Critical Path Testing Using Genetic Algorithm**. In: Bansal J., Das K., Nagar A., Deep K., Ojha A. (eds) Soft Computing for Problem Solving. *Advances in Intelligent Systems and Computing, vol 817. Springer, Singapore.*
9. S. S. Skiena. (2008). The Algorithm Design Manual Second Edition. https://doi.org/10.1007/978-1-84800-070-4
10. Gurari, E. (1999). http://web.cse.ohio-state.edu/~gurari/course/cis680/cis680Ch19.html - backtracking algorithms.
11. S. Das, D. Dereniowski & U. Przemyslaw. **Energy Constrained Depth First Search. (2017)**
12. Kirupa. (2006). kirupa.com - Depth First and Breadth First Search: Page 1.https://www.kirupa.com/developer/actionscript/depth_breadth_search.htm
13. P. Prakash, A. Sri & K. Rao. **R Data Structures and Algorithms Increase speed and performance of your applications with efficient data structures and algorithms.** *www.packtpub.com (2016).*
14. S. Hamimoune and B. Falah. **Mutation testing techniques: A comparative study**, *2016 International Conference on Engineering & MIS (ICEMIS).*
15. B. Srisura & A. Lawanna. **False test case selection: Improvement of regression testing approach**. *2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON).*
16. K. Vivekanandan, T. Megala and P. Chandini. **Automatic generation of basis test path using clonal selection algorithm**. *2016 International Conference on Information Communication and Embedded Systems (ICICES), Chennai, 2016, pp. 1-4, doi: 10.1109/ICICES.2016.7518907.*
17. O. Banias. **Dynamic programming optimization algorithm applied in test case selection.** *2018 International Symposium on Electronics and Telecommunications (ISETC).*
18. D.B. Mishra, S. Bilgaiyan, R. Mishra, A.A. Acharya and S. Mishra. **A Review of Random Test Case Generation using Genetic Algorithm.** *Indian Journal of Science and Technology, 10(30),2017.*
19. I.-A. Salihu, R. Ibrahim, B. S. Ahmed, K. Z. Zamli, & A. Usman. **AMOGA: A Static-Dynamic Model Generation Strategy for Mobile Apps Testing**. *IEEE Access, 1–1. doi:10.1109/access.2019.2895504.2019.*
20. S.W.G. AbuSalim, R. Ibrahim, J.A. Wahab. **Comparative Analysis between Selective and Non-Selective Mutation Techniques.** *International*

Journal of Emerging Trends in Engineering Research. Vol 8., No 4., April 2020, pp 1103-1110. https:://doi.org/10.30534/ijeter/2020/25842020

21. R. Ibrahim, M. Ahmed, R. Nayak, & S. Jamel. **Reducing redundancy of test cases generation using code smell detection and refactoring.** *Journal of King Saud University - Computer and Information Sciences. Vol 32, Issue 3, March 2020, pp367-374, https:://doi.org/10.1016/j.ksuci.2018.06.005*
22. P. Brereton, B.A. Kitchenham, D. Budgen, M. Turner, M. Khalil. **Lessons from applying the systematic literature review process within the software engineering domain.** *Journal of systems and software 80(4) (2007) 571–583.*
23. M.R. Penumala, J. Gonzalez-Sanchez. **Towards Embedding a Tutoring Companion in the Eclipse Integrated Development Environment.** *In: Nkambou R., Azevedo R., Vassileva J. (eds) Intelligent Tutoring Systems. ITS 2018. Lecture Notes in Computer Science, vol 10858. Springer.*
24. A. M. Erman, H. Fawareh. **Impact Cultural-Quality Factors on Successes and Failures Software System.** *International Journal of Emerging Trends in Engineering Research. Volume 8. No. 5, May 2020.*