A novel defect detection method for software requirements inspections

Bilal Alqudah[1], Laiali Almazaydeh[2], Reyad Alsalameen[2]
[1]Faculty of Engineering, Al-Hussein Bin Talal University, Ma'an, Jordan
[2]Faculty of Information Technology, Al-Hussein Bin Talal University, Ma'an, Jordan

Article Info	ABSTRACT
	The requirements form the basis for all software products. Apparently, the requirements are imprecisely stated when scattered between development teams. Therefore, software applications released with some bugs, missing functionalities, or loosely implemented requirements. In literature, a limited number of related works have been developed as a tool for software requirements inspections. This paper presents a methodology to verify that the system design fulfilled all functional requirements. The proposed approach contains three phases: requirements collection, facts collection, and matching algorithm. The feedback results provided enable analyst and developer to make a decision about the initial application release while taking on consideration missing requirements or over-designed requirements.

Corresponding Author:

Laiali Almazaydeh
Department of Software Engineering, Faculty of Information Technology, Al-Hussein Bin Talal University
Ma'an-71111, Jordan
Email: laiali.almazaydeh@ahu.edu.jo

1. INTRODUCTION

Software engineering is defined as the application of a standardized, structured, and thorough approach to the development process of the software in a rigorous way [1]. The process encompasses the entire range of activities, from initial customer inception to software production and maintenance. The engineering approach is the activity of envisioning and realizing valuable new functions with sufficient and justifiable confidence that the resulting software will have all the critical quality attributes that are necessary for the software to be a success. Therefore, as the end of software engineering is a streamlined and reliable software product, the software should be engineered correctly using the intersection between requirements, architecture, and project management, and all these essential concepts that have to go into the software engineering mix [2].

The intended software product is developed using structured sequences of stages in software engineering called software development life cycle (SDLC) [3]. The first stage in SDLC is requirement engineering since the requirements form the basis for all software products. Requirement engineering consists of a set of steps that are handled in an iterative process. The first step is elicitation which is the collection of requirements from stakeholders and other sources. The second is requirement analysis which involved the study and deeper understanding of the collective requirements. The third step is specification of requirements, in which the collective requirements are suitably represented, organized, and saved so that they can be shared. Once the requirements have been specified, they can be validated to ensure that they are complete, consistent, no redundant and so on. Finally, the fifth step is requirements management which accounts for changes to requirements during the lifetime of the project [4]–[6].

The eventual artifact that comes out of requirements engineering process, is the software requirements specification (SRS) document [7]. Typically, the SRS document will end up containing the requirements which can be classified along two different axes. One axis is that of the user versus system requirements. User requirements are written in a natural language, and system requirements are written more from a developer's perspective. Another axis can be differentiated is called functional and non-functional requirements. Functional requirements indicate the services from the perspective of the functionality of the system, and non-functional requirements indicate a particular behavior of function of the system [8]–[10].

However, the requirements can range from a high-level abstract description of the system services to a precise mathematically formulated specification. The reason behind this wide range in the requirements definition is because, it can serve multiple purposes. The requirement itself can be used as a basis for a request for proposals (RFP), so this may be a basis for a bid or contract. Therefore, in principle, the requirements have two important characteristics; the first characteristic is completeness to avoid ambiguity, and the second characteristic is consistency to avoid any conflicts or contradictions in the description of the system facilities [11]–[14].

In fact, the requirements are imprecisely stated, since it is ambiguous for interpretation, so both the client and developer will look at the requirements from their own perspective. But the ambiguity and the imprecision with which it was laid out can create significant problem later [15], [16]. In addition, the need for rapid production and lowering costs force some companies to release the application with some bugs, missing functionalities, or loosely implemented requirements. Furthermore, the traditional SDLC methodologies cannot go over the implementation as one unit for large systems. In addition, most of current algorithms focus on providing feedback regarding analysis-implementation phases in stages.

In this paper, we propose an automated methodology to focus on functional requirements implementation in the final product regardless of software size eliminating the need for a large number of reviewers or quality assurance (QAs). The proposed methodology is quantitative; however, there is no specific acceptance ratio specified for all systems ahead. It can be used for many rounds of inspections with no additional costs. The provided feedback enables the analysist and developer to make a decision about the initial application release while taking on consideration missing requirements or over-designed requirements. Below we describe the relevant literature, several alternative defect detection methods which motivated our study, our research methodology, and our test cases, results, and conclusion.

2. RELATED WORKS

Until now, however, a limited number of related works have been developed as a tool for software requirements inspections. One of these some key related studies in [17] where a controlled experiment was applied to assess different defect detection methods for software requirements inspections. The different defect detection methods are ad hoc, checklist and scenario-based detection method. The experimental results showed that the defect detection rate was higher when using a Scenario-based detection method, in which each reviewer focus on particular class of defects, than either ad hoc or checklist methods.

The work in [18] was based on defining design errors to different classification, which are: inconsistencies, inefficiencies, ambiguities, and inflexibilities, in order to review these errors by reviewers according to their skills and knowledge. The purpose of this classification is to ensure that the reviewers will find as many errors as possible. The work approach in [19] defined some software metrics in the factors and discussed several software quality assurance models and some quality factors measure method. One of these software quality factors is completeness and correctness of requirements, where the software quality measure metric is requirement specification. Other work in [20] followed divide and conquer policy, by decomposition of the inspection into discrete steps, so that one inspection step can be carried out without detailed knowledge of the others. The work in [21] considered correctness, which indicates the ability of a system to perform according to defined specification as one of software quality assurance factors. Meyer [22] also defined a more software quality factors and classified these factors into technical groups. One of these groups is product-based factors. Product based factors are those factors that define the "properties of the resulting software, for example correctness, efficiency" [22]–[25]. Moreover, Meyer derived these quality characteristics from McCall's quality taxonomy model.

Apparently, based on the mentioned related works, a few inspection methods are partly effective as inspectors may not have an adequate understanding of the inspection process as they take shortcuts. Therefore, still, further research is needed to find more practical and effective ways of doing inspections. In this regard, our contribution is developing a new automated approach that is used, in one hand, for quantifying the ratio of implemented requirements and over-designed functionalities, in addition to identify the acceptance ratio that affect the initial product release. On the other hand, it will lower the cost of requirements review by being able to re-run the evaluation process many times with no extra cost or time.

3. PROPOSED METHOD

Information generated about any system can be classified into information produced in the analysis phase and information produced as a result of development. Each stage of system development has many details and sub tasks. In those stages a lot of material will be available through requirements elicitation in the form of text documents, images, and scanned documents. Those information and details might get ignored or forgotten when scattered between development teams.

After the system is built, requirements became facts of the system. Some facts are hidden in a form of functionalities; for example: "reports have to be sorted by employee name". To be able to verify that the system design fulfilled all functional requirements, the system will be verified against requirements gathered. However, some requirements can be hidden as explained before. To overcome that problem, we propose a method where requirements automatically gathered regarding the system from analysts and from developed system in pre-processing steps. Those steps are summarized into: 3.1. Requirements collection, 3.2. Facts collection, and 3.3. Matching algorithm.

Requirements collection stage involves information extraction, forwardly. Foreword collection means: collecting information from analysis documents, text data, and images. Where facts collection is represented by reverse collection, this process is initiated from the final product side, from code, scripts, and graphical user interface (GUI). The algorithm takes the available resources (text data and image data) through optical character recognition (OCR) to extract text. In the matching phase, collected information are joined in sets representing requirements for that screen in the system by identifying key words in the collected text.

The last step is processing facts and requirements by the matching algorithm. The matching algorithm takes the responsibility of producing two sets of results, one is the matching requirements and facts. The other is the set of information is requirements found in the documentation but not in the designed system. Both sets will be represented by a numerically as well. The following sections provide details for all steps.

3.1. Requirements collection

Figure 1 shows gathering requirements from analysis phase. In this stage information collected by firstly; parsing the repository of text files generated through the analysis phase and requirements elicitation. Secondly, all images, pictures, and scanned documents are converted to text through OCR. The text extracted from all sources clustered in a map where important words classified in a special table.

The documents then classified based on functionalities, a matching table is created for requirement, document pairs (r, d) as shown in Table 1. The goal of that table is to show how many documents are related to requirement specified. Another goal is to be able to identify documents that does not relate to any requirement. Those documents either 1) analyzed in a wrong way and some requirements have been ignored or 2) the document does not relate to any functionality and the functionality has been forgotten for sufficient analysis. Document significance metric: for each row in the table, the sum of ones represents how significant is the document to the system. The number is assigned to the document as a document weight (dw) as shown by (1).

$$dw_i = \sum_{req=0}^{n} \text{rd[req]} \tag{1}$$

Table 2 shows the document classification matrix. For instance, doc_{n-1} has no importance to the system, or the document was ignored by mistake. That document needs to be revised and fixed to fit in its correct location regarding the system. Where doc_n on the opposite, talked about almost every requirement in the system except for two of them. That document should be revised as well because it is either an executive summery and has no details about the system and its development, if so, then it must be removed from the analysis we are doing, or it is not a summary, but it is a document shows the interaction between system components. In both cases, zero value documents and very high significance documents must be revised or removed from the verification we are conducting.

Requirements significancy metric: the sum of each column in the requirements document (RD) table represents how many documents talked about that requirement. This metric presented as requirement weight (rw) and calculated as shown in (2).

$$rw_i = \sum_{doc=0}^{n} \text{rd[doc]} \tag{2}$$

No requirement can have the value of rw=0 at all. This means that the requirements elicitation process missed the requirement, or there are some missing documents. Otherwise, the requirement should be marked as missing requirement and reported back. As shown in Table 3, requirement Req_{n-1} is missing from the analysis phase or documents analyzing it is missing.
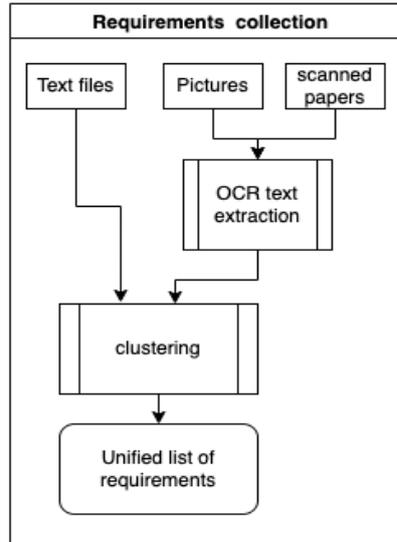
Figure 1. Gathering requirements from analysis phase

Table 1. Requirements document (RD) table

Req.\Doc.	Requirement1 [req1]	Requirement2 [req2]	…	Requirement n [req n]
Doc1	1	0	…	1
Doc2	0	1	…	1
…	…	…	…	…
Doc n-1	0	0	…	0
Doc n	1	1	…	1

Table 2. Document weight table

Document	dw
Doc1	2
Doc2	2
…	…
Doc n-1	0
Doc n	n-2

Table 3. Requirements weights matrix

Requirement	rw
Req1	2
Req2	2
…	…
Req n-1	0
Req n	3

3.2. Facts collection

From the other side of the system, the developed and running application, facts about the system collected and classified. Each function in the code, procedure, script with their corresponding interface is grouped in one cluster and named according to that feature. What is new is that a text file with what we called golden keys (gk) is created for each cluster. The gk set is used as a keyword set of what does the set of facts collected represents. The reason for that is the fact that some requirements such as functional requirements (font is bold, italic, the color is red with white borders) cannot be extracted easily from the design. To work around this problem, we created the golden-key set as shown in Figure 2. The gk set is better to be matched in name with the requirements specified in the RD table.

The collected list of facts now produces a filtered and clustered list of facts regarding each functionality. For example, assuming e-commerce system, the system is producing a report showed on the screen of customers sorted by last name. A list of facts regarding that report are (report ID, report date, issued by who, directed to whom, first, middle, and last name columns), all those facts will be clustered under one title called *report_by_name* cluster. A cluster of facts will be generated for each screen or window of the analyzed system. We will be referring to the window as a feature and the items of that window, text, and fields as facts. To connect the terminologies, a set of requirements and specifications in the analysis phase is called a requirement, a requirement has sub-fields for it. After the system is built and the code for that requirement is written we call it a feature and each feature have a set of facts. Table 4 shows an example of extracted feature-facts from some system of managing employees. Each feature (f) in a system will have set of facts $\{x\}$, a feature (i) represented as $F(i) = (i, \{x1, x2, x3..xn\})$.
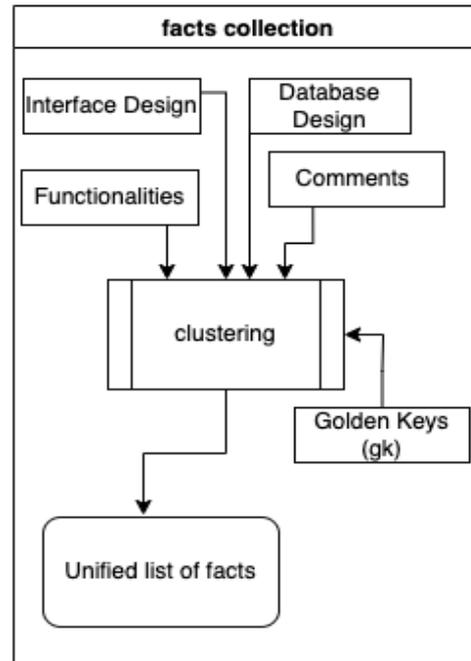
Figure 2. Facts collection and clustering with golden-keys

Table 4. Example of feature-facts

Feature f	Name (x)
Create_New_Employee	Full name
F1	Address
	…
	DOB

3.3. Matching algorithm

This is the quantitative component for user requirements versus GUI facts collected. The algorithm is built on the assumption of extracting information from images (representing forms, and paperwork) are provided in duplicate-free lists. To guarantee that there is no duplication found, facts are stored in hash sets that allows one copy of each fact in it. The results of extracted information are saved in lists. The other assumption is that the system interface, database has been established and the evaluation algorithm we are providing has access to the system and can run the same algorithms used previously to extract information form documentation and paperwork.

In the proposed algorithm 1, lines 5 and 6 gets the result of data mining and for images and text files add them in line 7 to a hash set where duplication will be eliminated automatically because of the feature that a set provides. This will allow us keep one copy of the feature extracted from the image or the text. In line 8, the while statement will get one feature from the user interface (UI) design and look for it in the hash set prepared in step 7. If the feature exists (line 9) in the hash set, this means that the feature from the UI has a match from the documentation and the images. For each feature found a match for, remove it from the UI features so the algorithm will not check it twice then move to the next UI feature to check if exist. This process will happen in line 11, if a match found between the set of features form the documentation, images and the UI, (m) will increase by 1 stating that a match found.

In line 15, the (v) factor will increase by the amount of information found in the hash set with no match from the UI design. The hash set will be cleared after that because we did the best with the information, we got the number of matches and the amount of miss. After clearing the set, the algorithm checks for the search depth factor specified. The depth factor specified by the algorithm user to indicate how many documents need to be mined if the acceptance ratio not found. This condition will help in stopping the algorithm from keep running indefinitely for large amount of data or if the ratio specified in line 22 not satisfied. In line 21, the loop will stop if the amount of information from the images and the documents still has no match comparing to the amount of information found in the UI less than the acceptance ratio specified. The algorithm will add {*m, v, currentDepth, i, j*} to an array of results and return them to the main function as a result for the match.

The following are the parameters which are used in algorithm 1:
- *Images*[*x*]   : A list of images of size x.
- *Text*[*y*]   : List of text files, documentation of a size y.
- acceptRatio   : The estimated accuracy level or matching level after which the system can be considered matching requirements.
- *depth*   : Until when the algorithm will keep running and asking for more data mining.
- *m*   : The matching percentage between the developed UI and the requirements.
- *v*   : The divergence between what is the in documentation and the UI (which is the result of the analysis).

The assumptions of the algorithm 1 are as follows:
- *ImgMine*(*Image*): Any selected data mining algorithm to extract information from images and return a set of features (we focus on the attributes related to text).
- *TxtMine*(*Text*)   : Any selected data mining algorithm to extract information form text files and documentations with ranking.

Algorithm 1. Matching algorithm
```
1    results [] match(images[x], text[y], acceptRatio, depth) {
2    m, v: will be returned in results.
3    currentDepth: int
4    do {
5          ImgMine(images[i])➔imgResults<>;
6          TxtMine(text[y]) ➔txtResults<>;
7          HashSet.add(imgResults), HashSet.add(txtResults)
8           While(UI.hasNext()) {
9            If(HashSet.contains(UI.getFeature(j)){
10                  m++;
11                  HashSet.delete(UI.getFeature(j))
12                  j++;
13          }
14            }
15   v+=HashSet.size()
16   HashSet.clear()
17   currentDepth++;
18   if(currDepth>depth) break;
19   i++;
20   y++;
21   }
22   while((v-m)/m)>acceptRatio)
23   results.add(m,v,currDepth,i,j)
24   return (results)
25   }
26   }
```

4. TEST CASES AND RESULTS

To show the principal of how the algorithm performs, it was implemented in a simple text editing software where features are limited as well as requirements. A software developer has been asked to write analysis for the assumed text editor ordered by testers. The developer came up with five text documents that explains the work of the text editor. As shown in Figure 3, the test case system we used has a unique 28 requirements extracted from the selected system. The analysis document extracted and found to contain 29 paragraphs. the expectations will be having some mismatch between analysis and system. The goal is to highlight the mismatch using the proposed algorithm.

First step is building the requirements-documents table to be able to identify the significancy of each document to requirements and vice versa. However, after building the tables, it can be identified that one of the documents found to contain functional requirements with no match to any functionality. Those functional requirements cover the coloring and fonts used. Then the algorithm tested using a clinic system where the available information is the system interfaces and the analysis files. The system was modified and part of it used to show the work of the proposed algorithm because of the non-disclosure agreement (NDA) policy for system owners and developers.

As shown in Table 5, documents covered the requirements for basic operations sorted ascending are: 1, 2, 3, 4..., where document 1 covered twice the requirements covered by the next document inline. Some requirements might be misrepresented or documented in a bad way, such as UTF8 and close document, in such cases those requirements need to be revised and the evaluation process must be run again. So, as shown in Figure 4, the algorithm found a matching.

Figure 5 shows the results gained from analyzing a test system built for a clinic. As the figure shows, the tested system has two screens each has a normalized 30 paragraphs of analysis and description. The requirements collected from the two GUIs contain 26 and 20 facts. In the first screen, the matching algorithm was able to find 5 facts that has no match in the analysis files and 9 key words that has no match in the running system. However, in the second system screen the algorithm found 15 key words in the analysis with no math in the running system and 5 facts or features in the running system with no mention in the analysis files. Those test cases show how the proposed algorithm was able to identify the mismatching between analysis and built systems. Those results will be useful feedback to QAs, analysist, and developers to minimize the rounds of code review and lower the cost of system development.

Figure 3. the initial set of requirements and facts sizes

Table 5. Document, requirement relevance

Requirement description	Req. Code	Doc1	Doc2	Doc3	Doc4	Doc5	$\sum rd$
text files	req1	1	0	0	0	0	1
plain text	req2	1	0	1	0	0	2
Unformatted	req3	1	0	0	0	0	1
Format	req4	1	1	1	1	0	4
UFT8	req5	0	0	0	0	0	0
new file	req6	1	0	0	0	0	1
open document	req7	0	0	1	0	0	1
close document	req8	0	0	0	0	0	0
add tables	req9	1	0	0	1	0	2
save as	req10	1	1	0	0	0	2
Save	req11	1	1	1	0	0	3
$\sum dw$		8	3	4	2	0	-------

Figure 4. Result of matching requirements from GUI and analysis files

Figure 5. Clinical system test case

5. CONCLUSION

This paper focused on validation the design fulfillment of user requirements as been provided by customer. However, we focused on the information presence in the design. As a future work; the paper can be improved by enhancing the algorithm by adding more sophisticated algorithms to match words and meanings such as "gender" selection box on drop box with the words (male/female) as substitute. This will improve the results but add more overhead to the cost (time). Another improvement can be integrating (user/designer) feedback to the algorithm to reduce the error ratio by stating whether the requirement has been fulfilled or not if the information present but not classified or matched by the algorithm.

REFERENCES

[1] R. L. Glass, I. Vessey, and V. Ramesh, "Research in software engineering: an analysis of the literature," *Information and Software Technology*, vol. 44, no. 8, pp. 491–506, Jun. 2002, doi: 10.1016/S0950-5849(02)00049-6.

[2] R. S. Pressman and B. Maxi, *Software engineering: a practitioner's approach*. McGraw Hill, 2005.

[3] B. Boehm, "A view of 20th and 21st century software engineering," in *Proceedings of the 28th international conference on Software engineering*, May 2006, pp. 12–29, doi: 10.1145/1134285.1134288.

[4] T. Rehman, M. N. A. Khan, and N. Riaz, "Analysis of requirement engineering processes, tools/techniques and methodologies," *International Journal of Information Technology and Computer Science*, vol. 5, no. 3, pp. 40–48, Feb. 2013, doi: 10.5815/ijitcs.2013.03.05.

[5] D. Pandey, U. Suman, and A. K. Ramani, "An effective requirement engineering process model for software development and requirements management," in *2010 International Conference on Advances in Recent Technologies in Communication and Computing*, Oct. 2010, pp. 287–291, doi: 10.1109/ARTCom.2010.24.

[6] D. Mishra, A. Mishra, and A. Yazici, "Successful requirement elicitation by combining requirement engineering techniques," in *2008 First International Conference on the Applications of Digital Information and Web Technologies (ICADIWT)*, Aug. 2008, pp. 258–263, doi: 10.1109/ICADIWT.2008.4664355.

[7] S. W. Ali, Q. A. Ahmed, and I. Shafi, "Process to enhance the quality of software requirement specification document," in *2018 International Conference on Engineering and Emerging Technologies (ICEET)*, Feb. 2018, pp. 1–7, doi: 10.1109/ICEET1.2018.8338619.

[8] B.-G. Lee, M.-S. Hwang, Y.-B. Lee, H.-J. Lee, J.-M. Baik, and C.-K. Lee, "Design and development of a standard guidance for software requirement specification," *Journal of KIISE: Software and applications*, vol. 36, no. 7, pp. 531–538, Apr. 2009, doi: 10.1145/3167132.3167268.

[9] V. Pekar, M. Felderer, and R. Breu, "Improvement methods for software requirement specifications: A mapping study," in *2014 9th International Conference on the Quality of Information and Communications Technology*, Sep. 2014, pp. 242–245, doi: 10.1109/QUATIC.2014.40.

[10] J. Medeiros, A. Vasconcelos, C. Silva, and M. Goulão, "Quality of software requirements specification in agile projects: A cross-case analysis of six companies," *Journal of Systems and Software*, vol. 142, pp. 171–194, Aug. 2018, doi: 10.1016/j.jss.2018.04.064.

[11] M. Ochodek and S. Kopczyńska, "Perceived importance of agile requirements engineering practices-A survey," *Journal of Systems and Software*, vol. 143, pp. 29–43, Sep. 2018, doi: 10.1016/j.jss.2018.05.012.

[12] D. Firesmith, "Prioritizing requirements," *The Journal of Object Technology*, vol. 3, no. 8, 2004, doi: 10.5381/jot.2004.3.8.c4.

[13] X. Lai, M. Xie, K.-C. Tan, and B. Yang, "Ranking of customer requirements in a competitive environment," *Computers and Industrial Engineering*, vol. 54, no. 2, pp. 202–214, Mar. 2008, doi: 10.1016/j.cie.2007.06.042.

[14] N. Bencomo, J. Whittle, P. Sawyer, A. Finkelstein, and E. Letier, "Requirements reflection," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*, May 2010, pp. 199–202, doi: 10.1145/1810295.1810329.

[15] M. Bano, "Addressing the challenges of requirements ambiguity: a review of empirical literature," in *2015 IEEE Fifth International Workshop on Empirical Requirements Engineering (EmpiRE)*, Aug. 2015, pp. 21–24, doi: 10.1109/EmpiRE.2015.7431303.

[16] D. M. Berry and E. Kamsties, "Ambiguity in requirements specification," in *Perspectives on Software Requirements*, Boston, MA: Springer US, 2004, pp. 7–44, doi: 10.1007/978-1-4615-0465-8_2.

[17] A. A. Porter and L. G. Votta, "An experiment to assess different defect detection methods for software requirements inspections," in *Proceedings of 16th International Conference on Software Engineering*, 1994, pp. 103–112, doi: 10.1109/ICSE.1994.296770.

[18] D. Parnas and D. M. Weiss, "Active design reviews: Principles and practices," *Journal of Systems and Software*, vol. 7, no. 4, pp. 259–265, Dec. 1987, doi: 10.1016/0164-1212(87)90025-2.

[19] M.-C. Lee, "Software quality factors and software quality metrics to enhance software quality assurance," *British Journal of Applied Science & Technology*, vol. 4, no. 21, pp. 3069–3095, Jan. 2014, doi: 10.9734/BJAST/2014/10548.

[20] D. L. Parnas and M. Lawford, "The role of inspection in software quality assurance," *IEEE Transactions on Software Engineering*, vol. 29, no. 8, pp. 674–676, Aug. 2003, doi: 10.1109/TSE.2003.1223642.

[21] E. Mnkandla and B. Dwolatzky, "Defining agile software quality assurance," in *2006 International Conference on Software Engineering Advances (ICSEA'06)*, Oct. 2006, pp. 36–36, doi: 10.1109/ICSEA.2006.261292.

[22] B. Meyer, "Applying 'design by contract'," *Computer*, vol. 25, no. 10, pp. 40–51, Oct. 1992, doi: 10.1109/2.161279.

[23] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: a survey," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 295–310, May 2004, doi: 10.1109/TSE.2004.9.

[24] Melville, Kraemer, and Gurbaxani, "Review: information technology and organizational performance: An integrative model of IT business value," *MIS Quarterly*, vol. 28, no. 2, pp. 283–322, 2004, doi: 10.2307/25148636.

[25] S. Yadav, "Analysis and assessment of existing software quality models to predict the reliability of component-based software," *International Journal of Emerging Trends in Engineering Research*, vol. 8, no. 6, pp. 2824–2840, Jun. 2020, doi: 10.30534/ijeter/2020/96862020.

BIOGRAPHIES OF AUTHORS

Bilal Alqudah received his doctorate degree in Computer Security and Privacy Protection from the Bobby B. Lyle College of Engineering, Southern Methodist University in USA in 2015. He is currently an assistant professor of Computer Security and Privacy Protection at the college of Engineering at Al-Hussein Bin Talal University, Jordan. Dr. Alqudah has held many local and international training seminars and conferences in his field of specialization. Dr. Alqudah focuses in computer security and privacy research, electronic medical records, and access controlling, in addition to other areas of interest. He can be contacted at email: alqu-dah@ahu.edu.jo.

Laiali Almazaydeh received her doctorate degree in Computer Science and Engineering from University of Bridgeport in USA in 2013, specializing in human computer interaction. She is currently a full professor and the dean of Faculty of Information Technology, Al-Hussein Bin Talal University, Jordan. Laiali has published more than sixty research papers in various international journals and conferences proceedings, her research interests include human computer interaction, pattern recognition, and computer security. She received best paper awards in 3 conferences, ASEE 2012, ASEE 2013 and ICUMT 2016. Recently she has been awarded two postdoc scholarships from European Union Commission and Jordanian-American Fulbright Commission. She can be contacted at email: laiali.almazaydeh@ahu.edu.jo.

Reyad Alsalameen received his Ph.D. in Software Engineering from University of Salford, UK in 2016. He is currently an assistant professor and the vice dean of Faculty of Information Technology, Al-Hussein Bin Talal University, Jordan. His current research interest includes fault-tolerant systems, software operation and maintenance, E-learning, and artificial Intelligence (AI) applications. He can be contacted at email: reyad.m.salameen@ahu.edu.jo.