Survey on Software Changes: Reasons and Remedies

Ibrahim Assi[1], Rami Tailakh[2], and Abdelsalam Sayyad[1]
[1]Joint Master in Software Engineering, Birzeit University, Palestine
[2]Mashvisor Real Estate Advisory, Palestine

Abstract: *Software systems play a key role in most businesses nowadays. Building robust, reliable and scalable software systems require going through a software production cycle (or process). However, it has been noticed that software systems are subjected to changes, whether those amendments are important or not. Those changes to software systems are viewed as a considerable issue in software engineering; they are considered as a burden and cost a lot, especially in cases such as enterprises and large-scale software systems. This study aims to identify the reasons that cause software changes and suggest remedies for those reasons. We survey the opinions of experts such as technical managers, team leaders, and senior developers. We collected 81 responses to our questionnaire, which aimed to establish common software development practices in the local industry. We also conducted 16 semi-structured interviews targeting the most senior experts, in which we directly discussed the reasons and remedies for software changes. Our results highlight the most influential reasons that cause changes to software systems, such as changes to user requirements, requests for new features, software development methodology, solving bugs, refactoring, and weak user experience design. Most importantly, the study solicited solutions that can reduce the need for software changes.*

Keywords: *Software changes, software maintenance, empirical study, survey, questionnaire, interviews.*

Received May 12, 2019; accepted April 8, 2020
https://doi.org/10.34028/iajit/18/2/14

1. Introduction

Over the past decades, a huge number of software systems were developed, and have been playing a key role in most businesses. However, research and studies significantly focused on the software properties that are being planned to develop rather than the maintainability of software systems that are already running [22].

Experienced software engineers believe that software changes are inevitable, it is a recurring task in both development and maintenance phases of the Software Development Life Cycle (SDLC). Therefore, more concentration should be given to the software properties such as changeability, complexity, and interoperability. Researchers look into ways to make software changes safer, easier and less costly [22].

Software maintenance is viewed as a key process in software engineering. This process consumes considerably more resources and efforts than the software development phase itself. Some studies claimed that the cost is between 40 and 67 % of the total cost of the system life cycle. Furthermore, many organizations use the majority of the total software budget on maintenance. As a result, researches should center on managing maintenance costs, and how it could be balanced with benefits. There is also another problem, which is how to estimate the ongoing efforts that will be spent on maintenance. According to one estimate, software maintenance constitutes 50% (or more) of the difference between the expected and ongoing efforts [6].

Another significant issue affecting software changes is software testing. In the software development process, mistakes are highly likely and are regularly committed by software developers. This necessarily requires to properly and thoroughly test software systems. Interestingly, software testing could reduce (or prevent) any future software changes. Majchrzak [14] states that “to program testing is to try to make it fail”. By testing software, potential faults and bugs, that are one of the causes of software changes, it could be reduced.

Nonetheless, there is no silver bullet. According to Brooks [5], software changes cannot be ultimately prohibited, and so, it is highly demanded to identify reasons for software changes. It is also required to understand why these changes took place; to reduce (or avoid) changes as much as possible.

In this work, we present the results of a two-part survey. In the first part, a questionnaire targeted senior software developers and managers from the Palestinian software industry, from which we received 81 responses. We asked questions that measured the frequency of software life cycle activities, to establish a baseline of the way the local industry operated as compared to the global standards. In the second part, we interviewed 16 of the participants in the questionnaire to specifically explore the research questions and to identify their opinions on possible remedies that can reduce software changes.

2. Research Questions

We designed our questionnaire and interview questions to be able to address the following research questions:

- RQ1: Does applying the standard methodology for the SDLC affect software changes?
- RQ2: Does gathering and changing user requirements affect software changes?
- RQ3: Does the experience of software developers affect software changes?
- RQ4: Does bug solving cause software changes?
- RQ5: Does upgrading of the running environment and development framework result in software changes?
- RQ6: Do code refactoring and non-functional requirements result in changes to software systems?
- RQ7: Does adding new features impact software changes?
- RQ8: Does reducing resource usage cause software changes?
- RQ9: Does the lack of HCI skills affect software changes?

3. Paper Organization

The remainder of the paper is organized as follows. Section 4 discusses related work and how we benefited from it in our survey. Section 5 explains the survey's protocol, participants, and data collection. Section 6 and section 7 details the results of the survey and discusses their implications. Section 8 outlines potential threats to validity and how we mitigate them. Finally, section 9 summarizes our conclusions and recommendations.

4. Related Work

Our review of software engineering papers, journals and books revealed considerable research concentrated on new software systems. However, there have been few studies that discussed the reasons for software changes in running software systems, whereas other studies illustrated software maintenance.

Some literature studies are concerned with data; such as developer's notes and code changes captured from version controls software. For instance, the researchers, in [25], studied the comments provided by code reviewers identifying defects in the submitted code via pull requests. As well known, in source safe software systems, pull requests are used to tell other software engineers about changes pushed made. In this research, the authors introduce deep learning algorithms in addition to utilizing developer's reviews. The target of this proposal is to find the best code change given a review; by studying the historical useful reviews. It is stated that this model helped minimize the workload that the developers suffer; for example, by providing review recommendations without humans interference. In other words, the model can help developers correct their code as soon as possible, thereby, reducing the time between each revision of code changes.

Similarily, Uqaili and Ahsan [27], they propose a model built employing Machine Learning algorithms in an attempt to predict software defects. Their model has shown accuracy metrics of precision and recall with above 90%. The desired goal of this model is to give prioritization to what changes are critical to be performed first. However, this research does not show what reasons that have caused these changes.

While other studies concerned the changes with regards to the platform. In [28], for instance, the study has addressed the influence of the frequency of changes, and the co-evolution of source code changes in mobile and non-mobile platforms. The investigation employed statistical and regression models to explain which factors affect the frequency of changes as well as find types of changes that frequently co-occur. The statistical analysis has shown that being mobile significantly impacts the frequency of changes. However, the study was conducted on Android systems concerning mobile platforms, while for the non-mobile platform, desktop and Web applications were considered. However, it is not stated what technologies and programming languages are used in building those applications.

Other efforts, in the literature, concern the effect of the development approach. In [3], a comparative analysis was conducted on the changes in the requirements. The comparison was made between In-House developed and global software development approaches; in an attempt to find what are common, and what is different in the two development approaches. The study was performed by conducting a questionnaire that was answered by industry practitioners. The comparative analysis and the survey resulted in a set of factors (challenges) that cause changes. These challenges range from cost and time estimation, document management, tracking of requirements, communication and coordination issues, knowledge management and sharing, and user involvement.

Mockus and Votta [18] had conducted a study, which focused on the textual description field that represents the software developer's remarks when applying their changes to the master source code. They used this feature to reduce the efforts of surveying the developers' opinions, in addition, to gain more accurate results to avoid any administrative or environmental influences, which could minimize developer's bias as well. Their studies concluded that there is the relevance between the size of comments added by developers, and the type of changes, on the one hand, and the required time on the other hand. However, the developer's remarks might be short, and sometimes they don't clearly explain the performed issues and countermeasures. The remarks feature isn't a mandatory field, whereby developers can apply their changes without filling up this field and results can be only considered as a start point.

Other studies were also concerned with analyzing textual data. For instance, Banker *et al*. [4] proposed a model that analyzed the errors stored in the log files. In their study, they made efforts to identify factors that affect the changes to the software. It was concluded that more frequent changes could cause more errors. Moreover, it was found that programmer experience matters for efficient system maintenance. That is, inexperienced programmers made significantly higher error rates than experienced programmers. Furthermore, the complexity of software systems may also affect error rates over time. Such noticed factors are more managerially controllable factors, which depend on the current management team, and may vary by changing the management team.

Rahman *et al*. [21], investigated whether clones of legacy code will make a difference that may cause defects to the software. Such defects will definitely be results of software modifications. However, they observed that the majority of software faults were not related to code clones. Moreover, the cloned code affected the non-cloned in the same way.

On the other hand, Sjøberg *et al.* [26] observed that the cloned code may require refactoring actions on the code to reduce such problems. As stated, refactoring is costly and it may introduce issues to the project. Most importantly, they concluded that minimizing the code size will reduce the number of modifications by minimizing code duplication, code arrangements, put it in line with global methods, and allow classes to be reused. Mondal *et al.* [19] stated that cloned code could be a problem, so any changes to the code necessarily require changing other similar software code. Therefore, they proposed an algorithm that may discover where the changes were made, which minimized the efforts of finding the cloned code.

Teamwork can be one of the key points for more productive projects, but Rastkar and Murphy [23] found that it is a challenge when several software developers work on the same project. A team member may individually make changes to the code which will result in conflicts among developers. It proposed techniques that generate human description in the form of document briefings, which may help developers to find out why a change happened and what are the actions that could be conducted. When this approach was experienced, the targeted developers suggested adding more technical information that gives more details about a specific piece of code.

Code changes are introduced in different forms. According to Ray *et al.* [24], modifications are frequent or repetitive. Unique modifications are different from the non-unique ones. However, developers commonly commit repetitive code than the broadening of unique changes. It is stated that such issues can be reduced by important approaches such as risk analysis, code reviews, and automated program repair. Likewise, Yan *et al.* [30] proposed to employ models that can help classify changes. It is also stated that such automatic classification could help managers to be well-informed, and so, it helps them make more appropriate decisions regarding these changes. For instance, they can have the decision of whether to increase quality assurance tasks such as software testing and code reviews. Kreutzer *et al.* [12], proposed a clustering model to detect the changes according to similar metrics. The model was used in automatic cluster groups of similar code modifications. It was interesting that they found that these groups of similar code changes made the bugs easily fixed. For example, similar changes may refer to a certain bug, such as the generated dataset by the clustering process, which may be helpful to be utilized for recommendation systems for software modifications processes. Code review could lead to better software quality. McIntosh *et al.* [16] state that code review is one of the important tasks in which another expert team member can review the code whether it is written in alignment with the best practices, standards (etc.,). In their research, they experimented on post-release defects, and they studied each fault with the coverage and participation of experts in code review. The most interesting thing that, they found there is an important connection between code quality with the code review extent, participation, and experience level of the experts who make reviews. Changes to code are inevitable, but this should not put the software at risk. Almasri *et al.* [2] proposed an approach that analyzes the impact of changes. This approach automatically measures the potential impact for a required change instead of depending only on the developer's experiences who provide support and maintenance for a given software system.

5. Research Methodology

The ultimate goal of this research is to identify the reasons behind the changes to software products in one hand, and what resolutions can help reduce (or avoid) such causes. Identifying these causes will be from the perspective of technical managers and software developers with senior level of experience. This research is comprised of a questionnaire and an interview. The questionnaire aims to discover the frequency of occurrence of software activities that are suspect as reasons for software changes, that were extracted from previous studies as mentioned in the Related Work section. While the qualitative approach through one-to-one interviews targeting experts in the field of software production and asks direct questions about the potential reasons of software changes, and then allows the participants to elaborate on their experience with those reasons and the remedies that they have used to mitigate them. Qualitative approaches are usually harder and time-consuming. [10]

5.1. Questionnaire

- *Protocol*: According to the previous studies (discussed in the related work section), a questionnaire has been designed with questions that could help quantify the rate of occurrence of the reasons for changes to software systems. The questionnaire contains 23 questions, in which each question (or set of questions) is to measure one of the expected reasons for software changes. The questions were selected to be direct and easily understood; that is, the audience could fill out the questionnaire in 10 minutes, or less. The questions cover important topic areas that measure the following factors: experience in terms of the number of years of experience in software development and management as well; standard methodology in SDLC; gathering requirements; software implementation (coding technologies and frameworks); software testing; support and maintenance; frequency of adding new features to software products; software refactoring; human interaction with software; and other areas related to software performance. As mentioned, the questionnaire almost covers the whole phases of SDLC on purpose measuring the most important reasons for software changes.

The design of this questionnaire was based on the guidelines provided by Kitchenham and Pfleeger [11]. Moreover, several references [1, 13, 20] were reviewed to thoroughly understand the software development life cycle, and specifically, the testing and maintenance phases. The questionnaire was reviewed and validated by different levels; a highly-experienced researcher with a Ph.D. degree, a Statistical Specialist, three senior developers, and a technical manager to check whether the questionnaire is direct and easily understand. In the beginning, and following Devore *et al*. [7], a sample of the audience was determined, which represented 10 % of the survey respondents; and to check whether the questions can be measured to help in achieving more accurate results. Moreover, the questionnaire included some questions that may result in contradictory answers, to help exclude random responses which may affect the results.

- *Participants*: The targeted population in this study is a group of highly-experienced developers and technical managers (see Figures 1 and 2.) working for medium-size businesses and enterprise levels in the local market. These companies and organizations have developed various applications in different forms; web, desktop, and mobile applications. To avoid bias and the possibility of divulging sensitive information about their companies, the questionnaire does not include any questions about organizations where the audience works. The participants were serious in their answers, that is, the questionnaire was filled up by 81 participants and only two responses were neglected because of their contradictory answers.

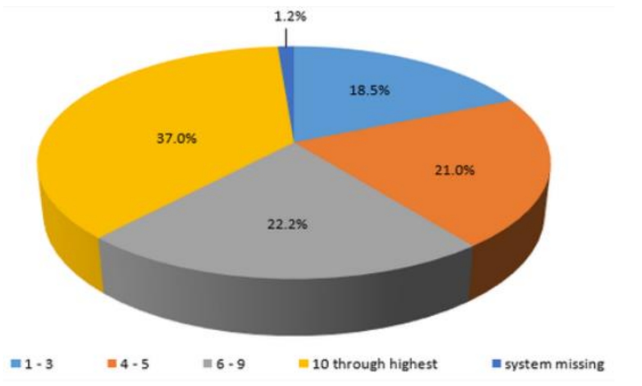

Figure 1. The Participants experience.

As shown in Figure 1., the highest percentage was for the years of experience “10 through highest”, which is 37 % of the total. While the period “6-9” years comes in the second place, at a rate of 22.2 %. Besides, the period of “4-5” years of experience comes in third place, with a rate of 21.0 %. Figure 2., Participants Experience Management, shows that the highest percentage was 56.8 % of “1-5” years of the respondent’s experiences as a manager/ team leader. While the second place was at a rate of 28.4 %, for “0” years. Moreover, the third place was at a rate of 12.3 % of the total, for the period “6-10” years.

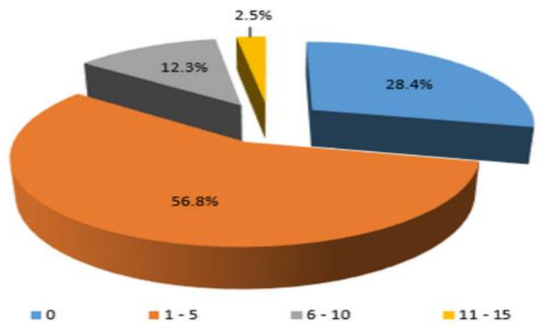

Figure 2. Participants experience management.

- *Data Collection*: The questionnaire survey was firstly published on Google forms. During the interviews, a hard copy of the questionnaire was asked to be filled, in case they did not fill the online copy. After that, the questionnaire results were filled out in the format of Comma-Separated Values (CSV) file, which is suitable for result analysis using Powerful Statistical

Software Platform (SPSS) CSV format which save data in a text file, the lines representing rows and each row separated by commas to representing columns [9]).

5.2. Interviews

- *Protocol*: The interviews took different forms such as Skype calls, phone calls, and one-to-one interviews. The interview was divided into three sections; the first one inspected their experiences, while the second section was to discuss reasons and resolutions that may decrease such changes. The third section included other issues that had not been presented during the interview. Other new issues and proposed remedies were also tackled by other participants were also been discussed. Concerning time, each interview took about 60-90 minutes on average.
- *Participants*: The targeted group was selected from different organizations and sections, working on various projects. In other words, they have had varied knowledge, roles, and experiences. More specifically, A total of 16 interviews were conducted; (6 technical managers, 4 team leaders, and 6 senior developers) with experiences of 4-20 years in software development, 4-15 years of experience in team leadership, in addition to technical management.
- *Data Collection*: All interviews were digitally recorded using personal smartphone voice recorders. Then the audio records were manually transcribed into texts. Finally, these interviews were summarized and categorized according to similar factors, opinions, and suggested solutions.

6. Results

In this section, we analyze the questionnaire results to measure the frequency of each question, which represents the expected reason for changes to software systems. Interview results are also analyzed to identify reasons and suggested remedies. Then the interview results are combined with the questionnaire results in the Discussion section.

- *Survey Outcomes*: In this section, a descriptive analysis of each question will be presented and discussed. Based on the descriptive illustrated in Table 2, it can be shown the number of participants, in addition to calculating means, standard deviations, and answers' percentages. The results shown in this table do not represent quantitative questions, but, it shows the answers to the scalable questions. All the questions of the questionnaire are mandatory; therefore, it was answered by the 81 participants. The percentage column in Table 2 shows the percentage of answering each question of the conducted questionnaire. This percentage is calculated by dividing the mean value over the maximum score; considering each question scored with a number. That is, it is scored from 1 to 5, then, multiplied by 100.

The Third Question (Q3) has the options: Never, Seldom, Sometimes, most of the time, and always; scored from 1 to 5 respectively. The percentage value of each option is calculated as the following:

- Never: $(\frac{1}{5})*100\% = 20\%$
- Seldom: $(\frac{2}{5})*100\% = 40\%$
- Sometimes: $(\frac{3}{5})*100\% = 60\%$
- Most of the time: $(\frac{4}{5})*100\% = 80\%$
- Always: $(\frac{5}{5})*100\% = 100\%$

to calculate percentage of question answer of Q3 by this equation: $(\frac{\text{Mean of questionanswer}}{\text{Maximumscore}})*100\%$ so the percentage of the Q3 is: $(\frac{3.44}{5})*100\% = 68.89\%$ Moreover, the questionnaire presents and discusses each research question, as seen in Table.1 that shows the relation between the research questions and questionnaire questions.

The first part of the questionnaire investigated the RQ1 (see Table 1). The Q3 in this part, the standard derivation is 1.245, and percentage was 68.89 % in average, which is close to the option “sometimes”, in which software development teams sometimes apply standard methodologies in SDLC.

The second part of the questionnaire concerns RQ2 (see Table 1) about gathering and changing the requirements.

The Fourth Question (Q4) the stranded derivation is 0.876, and the percentage was acknowledged by 64.20 %, which is close to the option “sometimes” of the detailed description of the customer's needs. Concerning the Fifth Question (Q5), the stranded derivation is 0.830 with participants responses are close to the option “sometimes” that was acknowledged by 55.06 % percentage. Question Six (Q6) in this part, the standard derivation is 0.846 with responses of 73.83 % which is close to the option “most of the time” of customer requirements changes after the beginning of implementation.

The developers' experience was measured in the third part of the prepared questionnaire, it answers the research question number 3, question RQ3 (see Table 1) about. In the Seventh Question (Q7), the standard derivation is 0.766 with respondents that are close to the option “senior”; which was acknowledged by 70.99 % where the majority of the participants are at the senior level.

The fourth section of the research questionnaire surveyed the audience's opinions about bugs solving that cause software changes, this answers the RQ4 (see

Table 1). The Tenth Question (Q10) in this part, the stranded derivation is 1.208 and the percentage was 61.23 % which is close to the option “sometimes”, in which 61.23 % apply a standard framework for testing. The standard derivation in (Q11) is 2.62, and 52.35 % of the participants are “Sometimes” use automated testing tools. (Q22) in this section fount that the stranded derivation is 0.679 and the percentage was acknowledged by 60.74 % of responded with the option “Sometimes” for the need for committing changes on code to enhance performance.

Table 1. Research questions and questionnaire questions.

Research Questions	Questionnaire Questions
RQ1: Does applying standard methodology for SDLC affect software changes?	Q3. Do you apply one of the standard methodologies; such as Agile, Scrum etc., for the software production life cycle?
RQ2: Does gathering and changing user requirements effect software changes?	Q4. Does the customer provide detailed description what their needs? Q5. Does the customer have documentation of the applied processes and procedures? Q6. Does the customer change the requirements after beginning of the implementation (coding) phase?
RQ3: Does the experience of software developers affect software changes?	Q7. What is your level of experience the technologies (Programming languages, development framework, database engine etc.) that will be employed for the implementation phase?
RQ4: Does bugs solving cause software changes?	Q10.Do you apply a framework for testing? Q11.Do you use automated testing tools? Q22.How often do you perform changes on code in order to enhance performance?
RQ5: Does upgrading of the running environment and development framework result of software changes?	Q13. Does a change of the environment; e.g.; upgrading/changing of the operating system, require changes on the software product?
RQ6: Do code refactoring and non-functional requirements result of changes to software systems?	Q8. Do any other software developers review your code? Q12. Does the software production team have QA members? Q15. How often do changes affect old features when applying new features? Q16. Do you often need alteration on software structure (Refactoring)?
RQ7: Does adding (a) new feature(s) impact software changes?	Q14. How often do customers request adding new features in one-month period of time after using a release?
RQ8: Does reducing resources usage cause software changes?	Q23. How often do you perform changes on code in order to reduce resources usage?
RQ9: Does the lack of HCI skills affect software changes?	Q17. What is the level of criticality of software products that you work on in general? Q19. Do software products you work on require training in general? Q20. Do you provide user manuals of the software products that you work on in general? Q21. How often do users make mistakes due to misusing the software?

Table 2. The descriptive statistics.

Question	N	Mean	Std. Deviation
Q3. Do you apply one of the standard methodologies; such as Agile, Scrum etc., for the software production life cycle?	81	3.44	1.245
Q4. Does the customer provide detailed description what their needs?	81	3.21	.876
Q5. Does the customer have documentation of the applied processes and procedures?	81	2.75	.830
Q6. Does the customer change the requirements after beginning of the implementation (coding) phase?	81	3.69	.846
Q7. What is your level of experience the technologies that will be employed for the implementation phase?	81	2.84	.766
Q8. Do any other software developers review your code?	81	2.89	1.118
Q10. Do you apply a framework for testing?	81	3.06	1.208
Q11. Do you use automated testing tools?	81	2.62	1.309
Q12. Does the software production team have QA members?	81	3.41	1.330
Q13. Does a change of the environment; e.g.; upgrading/changing of the operating system, require changes on the software product?	81	2.51	.853
Q14. How often do customers request adding new features in one-month period of time after using a release?	81	3.33	.908
Q15. How often do changes affect old features when applying new features?	81	2.89	.775
Q16. Do you often need alteration on software structure (Refactoring)? Note: Refactoring is the act of improving design without changing its behavior	81	2.88	.678
Q17. What is the level of criticality of software products that you work on in general?	81	2.35	.636
Q19. Do software products you work on require training in general?	81	3.48	.937
Q20. Do you provide user manuals of the software products that you work on in general?	81	3.25	1.210
Q21. How often do users make mistakes due to misusing the software?	81	2.89	.707
Q22. How often do you perform changes on code in order to enhance performance?	81	3.04	.679
Q23. How often do you perform changes on code in order to reduce resources usage?	81	2.69	.769

The fifth section of the questionnaire regarding the RQ5 (see Table.1). The standard derivation in (Q13) is 0.853, while 50.12 % of the answers were “Seldom” because of the software changes that cause upgrading/changing the operating system. The code refactoring and non-functional requirements are results of software changes that were measured in the sixth section, which answers RQ6 (see Table.1). In (Q8), the stranded derivation is 2.89, and 57.78 % of participants chose the option “Sometimes” regarding the of code review. Regarding (Q12), the stranded derivation is 1.330 with a percentage of 68.15 % answers chose the option “Sometimes” that there are Quality Assurance (QA) members in the development team (a QA in the software production process refers to the responsibility for ensuring that software meets the established standards set by the company or business) [16]. In (Q15), the standard derivation is 0.775 and percentage was 57.78%, which was close to the answer “Sometimes” regarding changes that affect old features when applying new features. Regards (Q16), the standard derivation is 0.678 and participants chose the option "Sometimes" with a percentage of 71.91 % for code refactoring.The seventh section answered RQ7 (Table 1) the standard derivation of (Q14) is 0.908 and the percentage was 66.67 %, which was close to the answer “Sometimes” for a customer's request to add new features in one month after using a release.While the eighth section related to RQ8 (Table 1) that measures the changes in software code to reduce resource usage. The standard derivation of (Q23) is 0.769, and 53.83 % of responses selected the option “Sometimes” for the need of making changes on software code to reduce resource usage. Finally, Human Interaction concerns the audience's surveys. In this section of the questionnaire, which answers the RQ9 (Table 1), it has been found the standard deviation of (Q17) is 0.636 and 46.91 % of the answers to this question were with the option “Moderate” for the criticality of software products. For (Q19), the standard derivation is 0.937, and the option “Sometimes” with a percentage of 69.63 % of answers that users sometimes require training of the way to use their software system. In (Q20), the standard derivation is 1.210 and the percentage was 64.94 % with the option “Sometimes” to offer user manuals of used software products. While the stranded derivation of (Q21) is 0.707, and the percentage was 57.78%, which was close to the answer “Sometimes” for users who make mistakes due to misusing the software, also that refers to the weakness of HCI.

6.1. Interview Outcomes

This section presents and discusses each individual interviewed to identify what factors could cause software changes, and, to what extent such factors can result in changes to software products. Besides, this section presents the suggested solutions by the highly experienced people who participated in this study interviews. Most importantly, each factor is presented in one of the research questions.

- *Applying Standard Methodology for SDLC (RQ1)*: Software development methodologies are very important for successful software products. As shown in Table 3, 14 participants considered that the “applying of standard SDLC methodology” is strongly affect reducing software changes. One other participant considered that it doesn't strongly affect. While another participant considered that applying a standard methodology somewhat can affect, during their usage, a customized methodology is applied to suit their special work environment and needs.

Table 3. Factors of software changes.

RQ	Effect	Remedies
RQ2 Gathering and changing user requirements	16: Strongly affects	14: multiple meetings with users 4: use mock-ups 4: use stories 4: use control panel 3: train devs in domain 3: study the user process most: changes in req are inevitable, just give more time.
RQ7 Add new feature	16: Strongly affects	4: improve requirements gathering
RQ1 Apply standard methodologies for SDLC	14: Strongly affects 1: Affects 1: Somewhat affects	8: apply SDLC to the letter 2: dedicate bug fixing iterations
RQ4 Bugs	13: Strongly affects 3: Affects	fairly common suggestions (see text)
RQ6 Refactoring	10: Strongly affects 3: Affects 3: Somewhat affects	5: involve reviewer and QA in dev team 5: hire experienced devs and train existing devs 5: build independent software services 2: use good design structure
RQ9 Lack of HCI skills	6: Strongly affects 6: Affects 4: Somewhat affects	9: employ UX designers 3: use mock-ups 3: run beta to get user feedback 3: reduce user clicks 2: reduce input fields, links, etc. 2: use validation messages
RQ3 Developer Experience	4: Strongly affects 12: Affects	8: involve reviewer and QA in dev team 4: use comments 4: expand business experience of devs. most: teams should mingle and loads should be balanced.
RQ8 Reduce resources	1: Affects 7: Somewhat affects 6: Neutral 2: Does not affect	
RQ5 Upgrade environment and development framework	5: Somewhat affects 7: Neutral 4: Does not affect	

In some businesses, standard methodologies are partially applied. Some companies hire their development teams to build their required software systems (in-house development). In some other cases, some of (small-business) companies hire one individual developer to build and maintain (a) software system(s). Moreover, there are challenges, to thoroughly applying standard methodologies, and face developers, especially when solving software bugs of previous versions. Additionally, new user's requirements and modifications may arise during the implementation phase affecting the good utilization of standard SDLC methodologies.

Interestingly, participants suggested solutions to resolve (or avoid) such mentioned issues. Eight of them recommended applying standard SDLC methodology with its all concepts. While two participants suggested customizing the SDLC methodology, according to their capabilities and needs. Two other participants stated that applying a standard methodology requires a management team to allocate a special period for solving bugs of the running version. Finally, one of the participants recommended being given enough time to apply the standard SDLC methodology.

- *Gathering and Changing User Requirements (RQ2)*: The collection of user needs and requirements is viewed as a ground for building systems that meet what is required. As shown in Table 3, all participants in consensus consider the "gathering of user requirements" is strongly important; a gathering of wrong requirements could cause changes to the software. Customers are not conscious enough about their needs, rather, they may only have a general idea without details, for example. After the implementation and the application of software, customers usually request changes to the software; in some cases, the changes may include cancellation of (recently) built feature(s).

During the requirements gathering phase, customers provide developers with documentation, which is sometimes not up-to-dated, does not describe their current processes and responsibilities. This may put the whole project at risk of failure, or it may cause expensive extensions. The changes to requirements are more challenging when users request their requirements on standard software (generic) used by many customers. In other cases, documentation reflects the vision of the manager while it does not reflect reality. For instance, some organizations hire temporary-contract consultants to develop such documentation. During the development of the documentation, these consultants usually interview managers and neglect the employees and workers who can describe their needs.

The participants suggest some solutions that can solve issues concerning the gathering of requirements. The majority of participants (14 participants) suggest holding multiple meetings with users until achieving clear user requirements. Three of the participants suggest to develop and train the development team in the work field. Three of the participants also suggest studying and analyzing the current user process to help users identify their needs well. Meanwhile, four participants recommend using mock-ups to validate and approve user requirements. Another four participants suggest to document requirements as a story, to be verified by users. The other four of the participants propose to add a module; control panel for example, within the software to enable or disable software features and services. Importantly, six of the participants ensure building scalable, easily maintained, and readable code as possible to reduce the effects of the changes to requirements. Three participants suggest setting a guideline for software-design architecture to meet any possible changes in requirements. One of the participants ensures that the existence of Quality Assurance (QA) members will help to monitor the gathering process in addition to review requirements. Seven of the participants state that there is no issue of any (important) changes if there will be extension to the delivery time according to the arose changes. Interestingly, most of the participants conclude that changes in user requirements are an inevitable process. However, any related issues should be reduced as possible to avoid expensive changes. It is worth mentioning that changes in requirements that could affect software architecture are costly as well.

- *Experiences of Developers (RQ3)*: The developers' experience is considered one of the key factors. Four of the participants, as illustrated in Table 3, state that the developers experience strongly play a vital role in reducing (or probably managing) the changes to software systems. On the other hand, 12 of the participants state that it may affect software changes. Software developers who have solid experience can better manage to reduce software changes, meanwhile, junior developers may cause changes (by increasing). It has also been discussed that senior developers may develop low-quality code as a result of the loads that may be assigned in a short period of time. Therefore, it is not only required to have highly experienced developers, but they also need enough time to achieve their tasks.

Development teams sometimes have different experiences that help developers, of beginner and intermediate levels, benefit from senior developers' experiences. In other words, teams should mingle to share knowledge and experience among team members. The presence of senior developers will also help validate code written by junior developers. Moreover, eight participants ensure involving code reviewers and QA members in the development team (from developers who have high experience). Two other participants propose holding regular meetings between developers to solve development problems. Four of the participants indicated that senior developers usually write comments on the written code that helps other team members in understanding their developed code. Concerning teams' development and training, four of the participants recommend widening developers' knowledge and experience in businesses and technologies as well.

- *Solving Bugs* (*RQ*4): During the software running phases, some bugs may result in changes to this software system. As presented in Table 3, 13 of the participants consider the "solving bugs" strongly affects software while it is running. While three participants do not consider it to cause considerable software changes. Moreover, solving the bugs process is viewed as changes to the software. It may also affect other components of the software. Besides, it will affect the action plan and result in creating confusion for developers who are already assigned to other tasks. This will impact users' confidence in the software, the company, or the development team. Solving bugs and testing software processes are also not given enough time. Furthermore, developers are not available; they are sometimes busy with other assignments, which may also introduce other issues that will require changes.

Participants in consensus agree that lacking software testing is one of the main causes of software bugs during systems running. One of the participants suggests improving the testing phase by including different techniques -not limited to- such as black-box testing, and automation testing in addition to designing different test cases and scenarios. Importantly, users should be involved in the testing phase. The majority of participants ensure that QA members should be involved in the testing phase. One of the participants recommends adding a pilot phase during the project to test and validate the software before production, while one of the important suggestions is to use machine learning in monitoring user behavior to help fins and identify software bugs.

- *Upgrade the Running Environment and Development Framework* (*RQ*5): From a different perspective, there might be changes to running environments. For instance, tracking software licenses and support, organizations may have to upgrade software such as operating systems. As shown in Table 3, five participants consider the "upgrade running environment and development framework" as a factor that may result in software changes, while the other seven participants were neutral against this factor. Four of the participants, on the other hand, consider that this factor requires software changes. Such a problem is inevitable, whereby an upgrade cannot be avoided.

- *Code Refactoring and Non-Functional Requirements* (*RQ*6): In software production, and for reasons such as the increase of concurrent users and delivered services, performance issues may appear. This requires enhancing software to resolve such non-functional requirements. The code refactoring is the process of software changes to improve internal structure without any change in external behavior [8]. The code refactoring is important to solving bugs and non-functional requirements such as the performance and to make the code maintainable and extensible. The refactoring code mostly occurred in legacy software and those who have bad designs. Furthermore, the needed code refactoring that was built by a new developer and not reviewed was coded by code reviewer or QA member.

As illustrated in Table 3, ten of the participants consider that the "code refactoring" has a strong impact on software changes, however, the other three participants consider that it affects the software, while the other three participants consider that it "somewhat" causes software changes.

Suggestions for such mentioned issues made five participants propose to involve QA members and code reviewers in the development team. Two other participants suggested building a strong design structure for software systems, in which they will be scalable and maintainable without the need for refactoring. Back to the developer experience factor, five participants ensured that hiring senior developers and experts in addition to keeping team members trained with the needed skills reduce (or avoid) such an issue. Interestingly, five of the participants recommended building each component of the software as an independent service that will reduce dependability between the software features. Another participant proposed to invest in building software services as microservices. Another participant suggested building the software features as plugins, in which any changes will be only performed on the plugins without the need to make any changes to the software that depends on such plugins. However, this issue cannot be ultimately avoided when there is interaction with hardware (and this hardware always changes).

- *Add New Feature* (*RQ*7): Adding new features is important in the software industry from a business perspective. A software system with more services could help attract more customers. In this research, all participants agree that adding new features is useful to software systems. However, this may have an effect on (some) old features causing considerable changes to software, which can be viewed as a problem. As seen in Table 3, all participants consider the "Add new feature" factor will strongly result in changes. Moreover, adding new features require one of the two activities; the code may either be a subject for extension or modification. For systems that are already running, adding new features will necessarily require software developers and testers to re-run testing cases on the running components to make sure that they were not affected. This is viewed as costly and cumbersome for developers and business owners as well.

Accordingly, two of the participants do not consider that a problem, rather, it is important to keep the sustainability of projects and businesses. While four of the participants see that some features can be viewed as new, that is, they were not clear during gathering requirements phase in the beginning.

- *Reduce Usage Resources* (*RQ*8): Running software systems requires some resources. These resources usually are limited; therefore, it should be taken into consideration how to properly manage and use these resources to the maximum extent. Consequently, the code sometimes needs to be modified to meet such limitations in resources.

On the other hand, two of the participants see that such a factor does not require changes. Seven of the participants consider it “somewhat” affecting, while 6 participants were neutral against this reason. One of the participants, on the other hand, sees that such a factor will cause changes to software systems. These readings are illustrated in Table 3. Interestingly, the available resources in the market are nowadays considered robust enough to handle new requirements.

As a suggestion, one of the participants stated that adding new resources is not challenging in their work environment. While two other participants proposed to change technologies to meet the available resources. One of the participants suggested that the code refactoring could also meet the available resources. While another participant recommended involving QA members and code reviewers in the development team to help build software code that meets the available resources.

- *Lack of HCI skills* (*RQ*9): The HCI is one of the areas that developers give attention in building (new) software systems. The HCI, in this study, refers to the Human-computer interaction, which is the field involves the design, evaluation, and implementation of interactive computing systems for human use. The HCI mainly focuses on end-user, user requirements, and context. By employing HCI, users are more involved in the design process; to achieve more user satisfaction. It also increases user productivity when using the software [15]. As a result, HCI concerns developing systems that are easy to use and meet user's needs. As shown in Table 3, six of the participants consider that lack of HCI skills will strongly result in software changes, and the other six participants see it effecting. While the other participants consider it “somewhat” causing changes. Any issues in usability do not necessarily refer to errors or bugs in software functionality; it is usually related to design problems that could affect user interaction with a software product. Such an issue arises when users are not able to easily understand and properly use specific software. This, then, requires to hold training sessions and provide user manuals to train users on how to use those software systems.

Two of the participants recommended building simple interface systems; with minimum input fields, links (etc.,). Other three participants proposed to reduce the number of user clicks that needs to achieve request service. Two other participants suggested enabling validation messages on the interface to avoid user's mistakes. While the other three participants ensured adding a pilot phase to the project and running a "beta" version of the software to enable users to give their feedback and opinions about the software. Interestingly, most of the participants (nine participants) suggested employing UX designers in the development teams; UX stands for user experience , which is viewed as a key factor for the success of software products, and so, software developers and designers should give great attention to UX in order to avoid any negative impressions when users use their products [17]. Again, three of the participants suggested providing users with mock-ups that help in building a more usable interface.

7. Discussions

This section discusses the combination of the interview outcomes and the questionnaire outcomes, in order to identify the reasons of software changes from among the frequently-occurring software life cycle practices in the local industry. The discussion is presented according to the research questions as follows:

- RQ1: Does applying standard methodology for SDLC effect on software changes? In the questionnaire, the majority's answer about applying the standard methodology for SDLC is (Q3) “Most of the time” and “Always”, which was acknowledged with 50 % of the participants. It can be noticed that most companies have applied standard methodologies for SDLC. In the interviews, 14 of 16 participants consider the methodology of applying standard SDLC strongly affects the minimizing of software changes. Therefore, applying a standard methodology for SDLC can be one of the key factors that could help manage and reduce changes to software products.
- RQ2: Does gathering and changing user requirements effect software changes? As shown in

the questionnaire (in questions; Q4 and Q5), it can be seen that there is a problem concerning the misunderstanding of users' requirements, as well as the changes to the requirements after the beginning of the implementation (coding) phase as shown in Q6. In the interviews, all participants, in consensus, consider the “gathering and changing of user requirements” is strongly important. As a result, the “gathering and changing of user's requirements” can be also viewed as one of the important causes of software changes.

- RQ3: Does the experience of software developers affect software changes? In the questionnaire; specifically, in question Q7, as well as in the interviews, the majority of the audience consider that the developers' experience is one of the key factors that affect causing software changes.
- RQ4: Does the bug solving cause software changes? In the questionnaire; especially the questions Q10, Q11 and Q22, which are related to software testing, the majority of the participants indicate that the weakness in software testing may cause bugs in the software system. This was also illustrated in interviews, in which most of the participants consider that the “solving bugs after release” strongly affect software while it is running. As a result, this factor can be also considered as one of the important reasons for software changes.
- RQ5: Does upgrading of the running environment and development framework result in software changes? This factor is not important compared to other reasons mentioned in the discussion. That is, the answer to Q13 in the questionnaire was between “sometimes” and “seldom.” This was also concluded in the interviews, and, therefore, this factor can be viewed as a weak factor.
- RQ6: Do code refactoring and non-functional requirements result in changes to software systems? Code refactoring and the non-functional requirements are related to each other. For instance, performance issues are usually resolved by code refactoring. The questionnaire conducted in this study includes four questions concerning such a factor; questions Q8, Q12, Q15, and Q16. According to the answers provided by the audience, it can be shown that code refactoring and non-functional requirements affect software changes. Most of the participants, in the interviews, also considered that this factor is an important reason that affects software changes.
- RQ7: Does adding new features impact software changes? The addition of a new feature is the most important factor. Adding new features can mean changes are committed to software products, which results in having more competitive software systems. This was indicated in the answers of Q14 of the questionnaire and the interviews as well.
- RQ8: Does reducing the usage of resources cause software changes? Sometimes, it is required to perform changes to reduce resource usages. In the questionnaire, most of the participants answered the question (Q23) in which most changes do not result from such causes. This was also indicated in the interviews, where most of the participants consider that this reason does not cause changes to software systems. Thus, this factor can be also viewed as a weak factor concerning software changes.
- RQ9: Does the lack of HCI skills result of software changes? This issue arises when users are not able to easily understand and properly use specific software (or module), which necessarily requires developers to perform changes to software products. The questionnaire of this research includes four questions (Q17, Q19, Q20, and Q21) related to HCI. It can be seen that the answers of the audience ensure that HCI results in important changes to software systems. In addition, this view was raised by most of the interviews in which most of the participants consider that HCI issues strongly affect software changes. This, as a result, requires developing the skills of HCI of software developers.

8. Potential Threats to Validity

This section will present and discuss the validity of the results achieved in this research. The validity check has been committed employing the methods outlined by Wohlin *et al*. [29]. This section is comprised of three evaluation parts: construct, conclusion and external validity.

- *Threats to Construct Validity*: Construct validity refers to the extent to which a piece of research actually investigates against the researcher's purpose of this study. The possible factors that may affect construct validity are presented as follows.

The opinions of developers and technical managers may be influenced by the interview discussion if it takes place before filling out the questionnaire. Therefore, we made sure that they filled out the questionnaire first. Also, during the one-on-one interview, they may also be influenced by the researcher; thus, the interviewer could not give any opinion (approval or rejection) but only a clarification about the question. The interviews conducted with software developers were held without the existence of their managers, in order to avoid any bias with their organizations or the applied methodologies.

- *Threats to Conclusion Validity*: This validation investigates the relationship between the conducted experiment and the outcomes, in which it examines the validity of the statistics discussed in the Results section against the hypothesis of the study.

The relationship between the technical managers and software developers with regards to software changes can lead to discrepancies in the opinions expressed by technical managers and developers. We cross-checked the outcomes of the quantitative and qualitative (questionnaire and interviews) analysis to reduce those differences.

- *Threats to External Validity*: The study targeted managers and senior developers in local market companies. Nevertheless, the tools, processes, and practices followed in the local market in Palestine are the same as other global markets. Therefore, the results of this research can still help global companies to understand and minimize software changes.

9. Conclusions and Recommendations

This work has addressed the problem of changes to software systems. In this research, efforts were made to identify reasons for software changes; by conducting a questionnaire and interviews that targeted technical managers, team leaders, and senior developers. The main results of this research support previous studies but also add insights into potential remedies that can mitigate reasons for software changes. As a conclusion, there are seven key reasons for software changes; applying the standard methodology for SDLC, gathering and changing user requirements, developer experience, bug solving, code refactoring and non-functional requirements, adding new features, and the lack of HCI skills. Suggested solutions for these issues also emerged during the interviews. It is also concluded that applying the standard methodology, for SDLC, reduces the impact of causes resulted from solving bugs and code refactoring. Moreover, enhancements in some development areas will reduce software modifications. These enhancements can be achieved by improving three main areas; development methodology, development teams, and the testing phase. For example, the development team should apply an appropriate development methodology that suits the nature of their software projects. They can also customize their development methodology if it is needed. Furthermore, developers should work as teams; thus, a team will be formed of different specialties with different experiences. For instance, a team can include back-end engineers, UX designers, software testers (etc.,). Finally, it is recommended to employ automation testing tools that help make the testing phase more efficient. User behavior can be investigated by using Machine Learning models on data, such as the data stored in web server access logs. Analyzing such logs may help proactively predict user needs. Applying such models could help to find the faults during the testing phase a well.

This research also surveyed expert opinions such as technical managers, team leaders, and senior developers. Future work may include controlled experiments to validate and test each of the concluded reasons and suggested solutions. Interestingly, this could drive to better identification of reasons for software changes.

References

[1] Al-Fedaghi S., “Conceptualizing Software Life Cycle,” *in Proceedings of Information Systems: Modeling, Development, and Integration*, Sydney, pp. 438-457, 2009.

[2] Almasri N., Tahat L., and Korel B., “Toward Automatically Quantifying the Impact of A Change in Systems,” *Software Quality Journal*, vol. 25, no. 3, pp. 60-640, 2017.

[3] Anwer S., Wen L., Wang Z., and Mahmood S., “Comparative Analysis of Requirement Change Management Challenges Between In-House and Global Software Development: Findings of Literature and Industry Survey,” *IEEE Access*, vol. 7, pp. 116585-116611, 2019.

[4] Banker R., Datar S., Kemerer C., and Zweig D., “Software Errors and Software Maintenance Management,” *Information Technology and Management*, vol. 3, no. 1, pp. 25-41, 2002.

[5] Brooks F., “No Silver Bullet,” *IEEE Computer*, vol. 20, no. 4, pp. 10-19, 1987.

[6] Carr M. and Wagner C., “A Study of Reasoning Processes in Software Maintenance Management” *Information Technology and Management*, vol. 3, no. 1, pp. 181-203, 2002.

[7] Devore J., Farnum N., and Doi J., “*Applied Statistics for Engineers and Scientists*,” Nelson Education, 2013.

[8] Fowler M., “Refactoring: Improving the Design of Existing Code,” *in Proceedings of 11th European Conference*, Jyväskylä, pp. 1-163 1997.

[9] Hasan M., Altab H., Hosney J., Haider N., and Touhid B., “CSV-ANNOTATE: Generate Annotated Tables from CSV File” *in Proceedings of International Conference on Artificial Intelligence and Big Data*, Chengdu, pp. 71-75, 2018.

[10] Hughes C., Qualitative and Quantitative Approaches, http://tinyurl. com/bmztxp8, Last Visited , 2012.

[11] Kitchenham B. and Pfleeger S., “*Guide to Advanced Empirical Software Engineering*,” Springer link, 2008.

[12] Kreutzer P., Dotzler G., Ring M., Eskofier B., and Philippsen M., “Automatic Clustering of Code Changes,” *in Proceedings of the 13th International Conference on Mining Software Repositories*, Austin, pp. 61-72, 2016.

[13] Langer M., *System Development Life Cycle (Sdlc)*, Springer Link , 2008.

[14] Majchrzak T., *Improving Software Testing: Technical and Organizational Developments*, Springer Science and Business Media, 2012.

[15] Mazumder F. and Das U., "Usability Guidelines for Usable User Interface," *International Journal of Research in Engineering and Technology*, vol. 3, no. 9, pp. 79-82, 2014.

[16] McIntosh S., Kamei Y., Adams B., and Hassan A., "An Empirical Study of the Impact of Modern Code Review Practices on Software Quality," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2146-2189, 2016.

[17] Minge M., Thüring M., Wagner I., and Kuhr C., "The meCUE Questionnaire: A Modular Tool for Measuring User Experience" *in Proceedings of Advances in Ergonomics Modeling, Usability and Special Populations*, pp. 115-128, 2017

[18] Mockus A. and Votta L., "Identifying Reasonsfor Software Changes Using Historic Databases," *in Proceedings of International Conference on Software Maintenance*, San Jose, pp. 120-130, 2000.

[19] Mondal M., Roy C., and Schneider K., "Acomparative Study on the Intensity and harmful-ness of Late Propagation in Near-Miss Code Clones," *Software Quality Journal*, vol. 24, no. 4, pp. 883-915, 2016.

[20] Nurmuliani N., Zowghi D., and Fowell S., "Analysis of Requirements Volatility during Software Development Life Cycle," *in Proceeding of Australian Software Engineering Conference*, Australian, pp. 28-37, 2004.

[21] Rahman F., Bird C., and Devanbu P., "Clones: What is that Smell?" *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 503-530, 2012.

[22] Rajlich V., "Software Change and Evolution" *in Proceedings of International Conference on Current Trends in Theory and Practice of Computer Science*, Milovy, pp. 189-202. 1999.

[23] Rastkar S. and Murphy G., "Why Did This Code Change?" *in Proceedings of the 35th International Conference on Software Engineering*, San Francisco, pp. 1193-1196, 2013.

[24] Ray B., Nagappan M., Bird C., Nagappan N., and Zimmermann T., "The Uniqueness of Changes: Characteristics and Applications," *in Proceedings of the 12th Working Conference on Mining Software Repositories*, Florence, pp. 34-44, 2015.

[25] Siow J., Gao C., Fan L., Chen S., and Liu Y., "CORE: Automating Review Recommendation for Code Changes," *in Proceedings of 27th International Conference on Software Analysis, Evolution and Reengineering*, London, pp. 284-295, 2020.

[26] Sjøberg D., Yamashita A., Anda B., Mockus A., and Dybå T., "Quantifying the Effectof Code Smells on Maintenance Effort," *IEEE Transactions on Software Engineering*, vol. 39, no. 8, pp. 1144-1156, 2013.

[27] Uqaili I. and Ahsan S., "Machine Learning Based Prediction of Complex Bugs in Source Code," *The International Arab Journal of Information Technology*, vol. 17, no. 1, pp.26-37, 2020.

[28] Viggiato M., Oliveira J., Figueiredo E., Jamshidi P., and Kästner C., "How Do Code Changes Evolve in Different Platforms? A Mining-based Investigation," *in Proceeding of IEEE International Conference on Software Maintenance and Evolution*, Cleveland, pp. 218-222, 2019.

[29] Wohlin C., Runeson P., Höst M., Ohlsson M., Regnell B., and Wesslén A., *Experimentation in Software Engineering*, Springer Science and Business Media, 2012.

[30] Yan M., Fu Y., Zhang X., Yang D., Xu L., and Kymer J., "Automatically Classifying Software Changes via Discriminative Topic Model: Supporting Multicategory and Cross-Project," *Journal of Systems and Software*, vol. 113, pp. 296-308, 2016.

Ibrahim Assi is the Head of Technical Support Unit at AlQuds Open University. He holds a Masters degree in Software Engineering from Birzeit University, and a Bachelor degree in Computer Science from An-Najah National University. His research interests are in empirical software engineering and human computer interface.

Rami Tailakh is a Senior Software Developer and Data Science Practitioner at Mashvisor Real Estate Advisory company. He holds a Masters degree in Applied Computing and Information Technology from the University of Bedfordshire, and a Bachelor degree in Electronic Engineering from Al-Quds University. His research interests are in empirical software engineering, machine learning, and predictive analytics.

Abdelsalam Sayyad is an Assistant Professor in Computer Engineering at Birzeit University. He holds a Ph.D. from West Virginia University and an M.Sc. from the University of Maryland at College Park. His research interests are in search-based software engineering and empirical software engineering.