



Gamifault: A Gamified Motivational Approach to Make Software Debugging an Attractive Process

Shahrzad Sadat Mousavi Esfahani^a, Mojtaba Vahidi-Asl^{a,*}, Alireza Khalilian^b,
Parastoo Alikhani^c, Bardia Abhari^a

^aFaculty of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran.

^bDepartment of Computer Engineering, Parand Branch, Islamic Azad University, Tehran, Iran.

^cFaculty of Education and Psychology, Shahid Beheshti University, Tehran, Iran.

ARTICLE INFO.

Article history:

Received: 4 September 2022

Revised: 31 December 2022

Accepted: 15 February 2023

Published Online: 3 May 2023

Keywords:

Software Fault, Debugging, Fault Localization, Gamification.

ABSTRACT

The released software systems still involve some faults, for which debugging becomes necessary. On the one hand, manual software debugging remains an arduous, time-consuming, and expensive task. On the other hand, effective software debugging is organized around motivated and patient developers. In this paper, a novel approach, namely Gamifault, is provided to make debugging more attractive and enjoyable. Particularly, the objective of Gamifault is to make the developer more curious to proceed debugging, that is fault localization and program repair, enthusiastically. To achieve this objective, the concepts and potentials of gamification are adapted to the typical tasks of software debugging. In particular, Gamifault makes use of an existing fault localization technique to determine the likelihood to each statement may be faulty. Based on the likelihood, the developer then attempts to find the exact fault location and fix the fault. Next, Gamifault reacts to the developer with a gamified success rate. That is, it shows the number of test cases that have been passed on the modified program. This process is repeated until the program passes on every given test case. To evaluate Gamifault, a prototype web-based tool was implemented in Java that targets faulty software programs. Then, 16 developers were asked to employ gamified and non-gamified versions of the tool in their debugging activities on 46 subject programs taken from the Code4Bench suite of programs. Developers could successfully debug 7 and 95 faulty programs using the non-gamified and gamified tools, respectively. In addition, the gamified tool helped developers debug the faulty program in less than two minutes on average. These results suggest that Gamifault offers advantages over existing debugging systems.

* Corresponding author.

Email addresses: s.mousaviesfahani@mail.sbu.ac.ir (SH. Esfahani), mo_vahidi@sbu.ac.ir (M. Vahidi), akhalilian@piaau.ac.ir (A. Khalilian), p.alikhani@sbu.ac.ir (P. Alikhani),

b.abhari@mail.sbu.ac.ir (B. Abhari)

<https://dx.doi.org/10.22108/JCS.2023.135012.1107>

ISSN: 2322-4460



1 Introduction

In the last decade, humans have been increasingly relying on software systems for every task. This implies that software quality is a vital necessity. However, the quality can be adversely affected by the software faults that are inevitably and prevalently introduced during various activities of the software lifecycle [1, 2]. As a consequence, software debugging becomes indispensable to maintain the quality of the software system at a reliable level. However, manual debugging is still an arduous, daunting task. Automated techniques [3–5] could have mitigated some of the burdens involved with manual debugging. Nevertheless, the overall complexity of software faults [6] necessitates the developer much patience and motivation. As of today, an immense number of techniques and tools [7, 8] have been proposed to assist developers in various debugging tasks. However, less attention has been given to providing and maintaining developer motivation during debugging, in particular for fault localization [9, 10] and program repair [11, 12].

In recent years, an increasingly rapid growth in the advent and usage of computer games has emerged [13]. Computer games are widely used by a diverse spectrum of people as an effective means of enjoyment and entertainment. The wide prevalence of games has incentivized researchers and practitioners [14] to adopt game elements and mechanics in nongame situations. This new application of games is called gamification [15, 16]. As a result of a preliminary investigation, it was found that gamified debugging techniques, particularly fault localization, and program repair, have received less attention in the literature [17–20].

In this research, the authors seek to exploit the potential of gamification to make the debugging process more attractive and enjoyable. It is speculated that providing and maintaining some sort of attraction and enjoyment for developers can partly mitigate the burdens and difficulties during software debugging. Therefore, the objective of this research is to make the developer more curious to proceed debugging enthusiastically. Gamification can be a promising way to accomplish this objective. Consequently, this research is explored to find the answer to the following research questions. Which game elements can be applied for debugging? How can debugging be made a motivational and competitive task?

To find the answer to the mentioned questions, a novel approach, namely Gamifault, is developed to incorporate the elements and mechanics of computer games into an automatic fault localization (AFL) technique [21]. Figure 1 demonstrates the high-level architecture of Gamifault. In particular, Gamifault receives a faulty program and an associated test suite that

contains at least a failing test case. Then, it computes suspiciousness scores for statements based on an AFL technique. The scores reflect the likelihood of which statements may be faulty. The developer then investigates some of the faulty statements and considers modifying them. Next, Gamifault executes the modified program against the given test suite to determine which test cases are passed or failed.

On the one hand, reliance on test cases can lead to imprecise fault localization since test cases serve as partial specifications of programs [17]. On the other hand, fault localization is generally an unsolved problem due to the immense diversity of fault manifestations as well as limitations with current techniques [5, 10]. These factors can explain why fault-localization techniques do not always determine the precise fault location. This issue establishes a situation in which the developer should consider modifying many statements as potential fault locations. The mentioned occasion can be quite burdensome. To keep the developer's attention, Gamifault informs him/her of the progress and results of his/her modifications via some augmented capabilities such as the scoreboard and some other game-like elements. The overall process is therefore designed to retain the motivation of the developer and alleviate the burdens induced during the debugging. The ultimate goal of Gamifault is to improve the effectiveness of the debugging process and help remove more fractions of the faults, yielding high-quality software.

To evaluate Gamifault, a tool in Java language was implemented. In addition, a non-gamified version of the tool was provided to be compared to the gamified version, as a baseline. In the tool, Tarantula [21] was used, which is an effective AFL technique and can be simply adopted. Then, 46 faulty programs were chosen as experimental subjects from Code4Bench [22] suite of programs. Next, two separate groups of developers were asked to apply the gamified and nongamified versions, respectively. Using the gamified version, the developers could fix 95 faulty programs in 982 minutes, whereas in the non-gamified version, the developers could debug and fix 7 faulty programs in 254 minutes. These results provide some evidence and suggest that Gamifault can be used as an effective debugging assistance tool to improve the motivation of developers during the debugging process. In addition, an online survey was conducted on the participating developers. In the survey, a questionnaire was provided in which a research question was investigated as to whether the participants are satisfied when applying Gamifault. With a confidence degree of over 95 percent, Gamifault turns out to be significantly effective in the improvement of developer motivation.



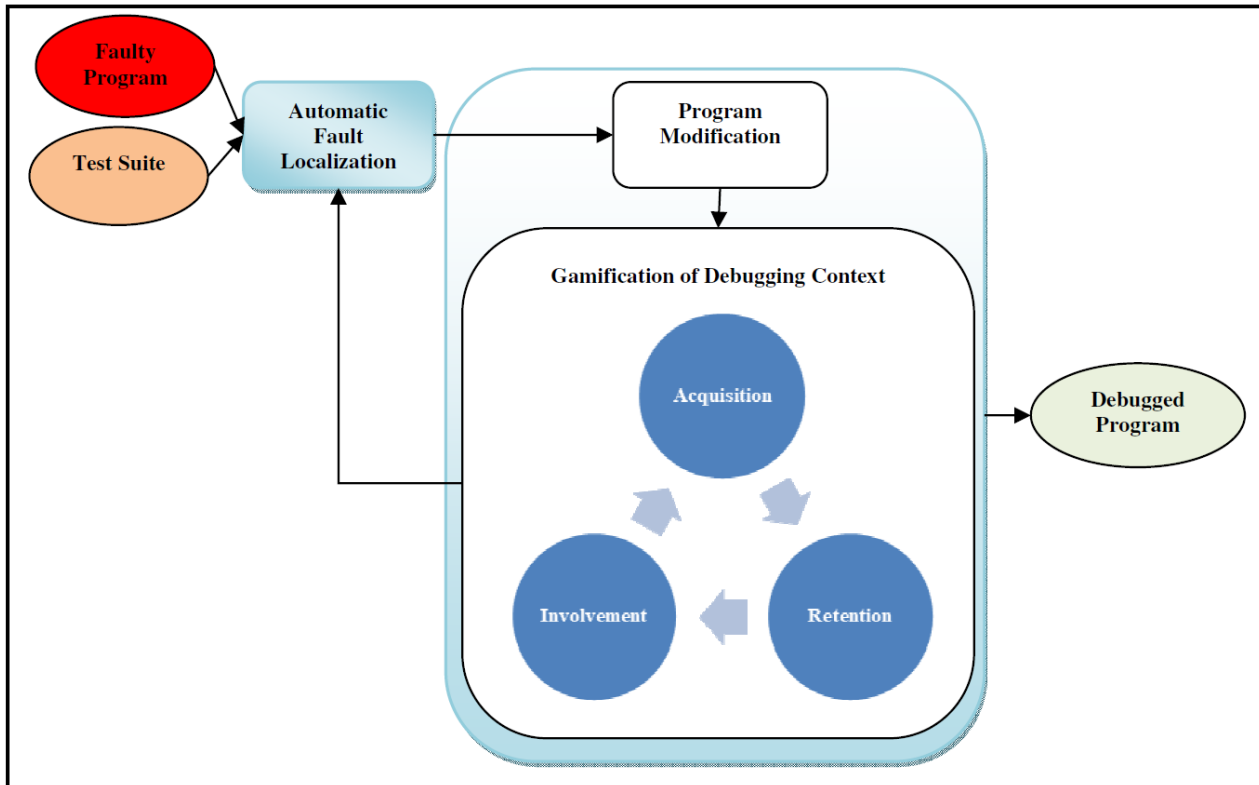


Figure 1. The High-Level Architecture of Gamifault.

The significance of Gamifault is to contribute to a practical problem faced by developers. The impact of Gamifault on researchers is to incentivize them to apply gamification to other software engineering tasks. For practitioners, it can reduce the vexation in the achievement of a higher quality software system.

In summary, the main contributions of this paper are as follows:

- Gamifault, an adaptation of game elements and mechanisms on fault localization and program repair
- Implementation of Gamifault in a prototype tool
- Empirical studies on the effectiveness of Gamifault for debugging several subject faulty programs
- A survey to investigate the satisfaction of developers using Gamifault

The remainder of this paper is structured as follows: In Section 2, fundamental concepts and a review of the related work are presented. In Section 3, the methodology of the proposed approach is elaborated. In Section 4, the steps that were followed to evaluate the new approach are articulated. The paper is concluded in Section 5.

2 Research Background

In this section, the fundamental concepts that are mentioned in the subsequent parts of the paper are elaborated on. Then, the major body of the literature that is related to the current study is reviewed. Finally, this section is concluded with lessons learned, the existing research gap, and the introduction to the new approach presented in this paper.

2.1 Fundamental Concepts

A couple of categories of concepts are required to grasp the ideas presented in this paper. The first category deals with software testing, debugging, and fault localization. The second category is associated with computer games and gamification.

Software Testing, Debugging, and Fault Localization Developers prevalently leverage software testing [23] to improve the quality of the programs. To achieve this, they construct a test suite, TS , and execute the given program p against every test case t in TS . A test case t often consists of the input data, along with the expected output. A test case t is said to be passing if the actual output of executing p against t is the same as the expected output of t . Otherwise, the actual and the expected outputs differ, and t is said to be failing against p .



Table 1. A Sample Code Regarding How Tarantula Works.

ID	Statement	Test Case						Tarantula	
		3, 3, 5	1, 2, 3	3, 2, 1	5, 5, 5	5, 3, 4	2, 1, 3	Score	Rank
	mid() { int x, y, z, m;								
1	read ("Enter 3 numbers:", x, y, z);	•	•	•	•	•	•	0.5	4
2	m = z;	•	•	•	•	•	•	0.5	4
3	if (y < z)	•	•	•	•	•	•	0.5	4
4	if (x < y)	•	•			•	•	0.63	3
5	m=y;		•					0.0	5
6	else if (x < z)	•				•	•	0.71	2
7	m = y; // Faulty statement	•					•	0.83	1
8	else			•	•			0.0	5
9	if (x < y)			•	•			0.0	5
10	m=y;			•				0.0	5
11	else if (x < z)				•			0.0	5
12	m = x;							0.0	5
13	print (x < z)	•	•	•	•	•	•	0.5	4
	}	Pass	Pass	Pass	Pass	Pass	Fail

Once t is found to fail on p , it turns out that p is faulty. To address this issue, the developer needs to debug p , which implies that the developer should find the exact location of the fault and proceeds by applying some modifications to p to fix the fault. The former task is known as fault localization whereas the latter task is called program repair. Currently, a proliferation of automatic techniques for fault localization [9, 20] and program repair [4, 12, 17] is presented. However, current automatic fault localization and automatic program repair (APR) techniques do not show efficacy for every kind of fault [10, 11]. There exists an infinite number of ways to implement an algorithm in a programming language. In addition, there exists an inordinate kind of fault where a certain fault can be manifested in various forms. As a consequence, for a given faulty program t and an associated test suite TS , an AFL technique may wrongly label a fault-free statement as faulty. Similarly, an APR technique may generate an incorrect patch. Hence, the debugging process still relies, to a large amount, on human intervention.

A popular and effective category of AFL techniques includes spectrum-based fault localization (SBFL) techniques [24, 25]. SBFL techniques execute the faulty program p against every test case t within the associated test suite TS . As a result of this task, the program spectra are collected. The program spectra for p and TS can be considered as a matrix $MAT(p)$, in which rows indicate test cases and columns indi-

cate program entities. Each entry at row t and column pe of $MAT(p)$ shows whether the execution of p against t covers (executes) pe . Program entities can be various elements in a program, from fine-grained elements, such as a simple statement, to coarse-grained elements such as a whole function. They can also include decision (branch) statements or definition-use pairs. To collect the program spectra, the given program p should be instrumented. That is, extra code should be added to the program to capture the coverage of the desired program entities at runtime.

Among the diverse collection of spectrum-based fault localization techniques, Tarantula [21] is highly known in the research literature as a baseline technique and has been widely used by practitioners. Tarantula is defined according to Equation 1:

$$Score_{Tarantula}(pe) = \frac{\frac{N_{CF}}{N_F}}{\frac{N_{CF}}{N_F} + \frac{N_{CS}}{N_S}} \quad (1)$$

where pe is a program entity, $Score_{Tarantula}$ denotes the suspiciousness score for pe , N_{CF} is the number of failing test cases covering pe , N_F is the total number of failing test cases, N_{CS} is the number of passing test cases covering pe , and N_S is the total number of passing test cases.

A spectrum-based fault localization technique executes a given program on the associated test suite and collects the program spectra. Then, it employs



a formula, by which it assigns a suspiciousness score to each program entity. The scores indicate the likelihood of a program entity being faulty. Next, the program entities are ranked (arranged) according to the scores where the top-rank program entity is typically the one with the highest score. The developer then inspects the program entities starting from the top-rank program entity. To illustrate how an SBFL works, consider the example in Table 1, which shows how Tarantula works on a sample program.

In the example, Tarantula has assigned the highest score, hence the top rank, to statement 7, which is the truly faulty code element. The developer starts investigating the faulty program from the top-ranked statement. For this single-line fault, the corrective modifications may not be trivial and the developer may attempt several rounds of modification. In addition, the example presents a substantially small program whose aim is to convey the functionality of SBFL techniques; it does not generalize to a large-scale, industrial software system where there exist thousands of test cases and hundreds of thousands of execution paths. Further, software systems may often involve multiple-line faults or multiple faults [19, 26], for which localization and repair necessitate much more effort. As a result, it is often the case where an AFL technique does not assign the highest score to the truly faulty program entity [27]. In practice, therefore, the debugging process turns out to be quite time-consuming and arduous.

Computer Games and Gamification. A computer game is a software system that is often used for entertainment. A computer game involves some rules that the player should conform to be able to proceed with the game and achieve a predefined objective. For example, in a war-like game, the objective may be to annihilate a supposed enemy, which in turn leads to the winning of the player [28, 29]. The game software system typically makes use of multimedia capabilities such as voice, animation, and video to attract the player. In addition, some sort of scoreboard is shown to highlight the progress in the game scenario and the objectives that are achieved. The scoreboard can also be considered a strong incentive for a player to further attempts the completion of the game.

Computer games consist of several categories, namely, entertainment games, serious games, simulation games, games with a purpose, and gamification. An entertainment game is just designed to make the player amused. A serious game is a software system that is designed to fulfill a certain goal rather than just entertain the player. For instance, training kindergarten kids can be a goal of some computer games [30]. In simulation games, game-design techniques are used to make a virtual environment for low-risk training,

that is, training in unreal situations. As an example, flight simulators are widely used during the training of pilots [31]. A game with a purpose is an application in which human computations are transformed into some sort of game [32]. The goal is to make a tiring task into an entertaining activity. Rings [28] is a game that helps developers generate high-quality test cases for effective software testing. Finally, gamification is defined to apply the elements and mechanisms of game software systems in non-game situations [14, 33–35]. The major aim is to make an attractive and incentive approach to keep the operator involved in the assigned process. In addition, the assigned process seems less arduous for the operator.

2.2 Related Work

Several studies are related to this research. In this section, the major relevant studies are reviewed in several categories. The aim is to identify the specific gap that this paper is intended to address.

Debugging Assistance. Researchers have developed an immense number of debugging assistance tools that are used in various practical situations. Brodie et al. [36] proposed a fixed recommendation method for recurrent faults. The method relies on a database of fault symptoms taken from stack calls. The design of the method was motivated by the huge costs that were incurred during the maintenance stage of the software lifecycle. Abraham and Erwig [37] presented a novel technique for debugging Microsoft Excel spreadsheets. The technique generates some recommendations and offers five top-ranked fix recommendations to the user. In some situations, the original programmer is not accessible, making the perception of the failure report so difficult. To mitigate this issue, Weimer [38] introduced a new technique that generates a proposed patch and a counterexample for a given failure report. These outputs can be used as the initial guidance for debugging the program. Ashok et al. [39] presented DebugAdvisor as a debugging assistance tool. The user needs to give a query as input, in which some information such as a memory dump is attached. The experiment on the tool in Microsoft has led to 78 percent of successful performance. BugFix is a debugging assistance tool that is introduced by Jeffrey et al. [40]. The tool leverages associate rule mining to learn appropriate fixes. Then, for a given fault, it recommends a prioritized list of potential fixes. Hartmann et al. [41] proposed HelpMeOut which is designed to offer fix suggestions that have been used by previous programmers to address similar issues. Kaleeswaran et al. [7] proposed MintHint that can be used to suggest appropriate hints to fix the fault. FIXBUGS is an extended version of FindBugs, which is developed by Barik et al. [42]. The proposed tool generates some



slow fixes, that is, step-by-step intermediate changes that can result in fixing the fault. In this way, the programmer is enabled to explore other possible modifications to fix the fault. This scheme is intended to establish a trade-off between manual and automatic debugging of programs.

Fault Localization. In the literature, there exist several categories of techniques that attempt to identify the most suspicious code regions or code elements that are highly likely to be responsible for the detected fault(s). They include slicing-based [43], state-based [9], machine learning-based [44], statistical-based [45], and SBFL techniques [46]. Among these categories, various large-scale empirical studies [5, 20, 26, 27, 46] have provided experimental confirmations in support of the effectiveness of SBFL techniques. Some of the SBFL techniques include Tarantula, Jaccard, Ochiai, Ample, Zoltar, and Sokal. SBFL techniques often follow the same approach, as mentioned by Tarantula. However, the difference among SBFL techniques just lies in the formula that they employ to achieve the suspiciousness scores of the code elements.

Despite the significant achievements, fault localization is generally an unsolved problem [10, 47]. That is, every fault localization technique shows experimental effectiveness in certain situations; neither of the techniques has proven efficacy for all situations. For large-scale software systems, a fault localization technique may label a large region of the code as potentially faulty code elements. Alternatively, it may often be the case that many code elements receive the same rank, possibly the top rank. These situations make the developer inspect a large amount of code. As a result, AFL techniques may practically turn out to show low effectiveness and the debugging process remains an arduous task [48].

Gamification for Software Engineering. Researchers in the field of software engineering have explored the application of computer games for various serious tasks. Arai et al. [49] have applied gamification to investigate the warnings of fault pattern tools. The study is aimed at motivating developers to remove the warnings issued by the FindBugs tool. Code Hunt is a web-based platform that is presented by Tillmann et al. [50] to help students write a program. The players (students) modify and evolve a program and the code hunt generates feedback as some sort of score. The feedback is achieved based on the number of test cases that the modified program can pass. Rings [28] is a gamified tool to help developers improve the process of software testing. Particularly, the tool is aimed at generating effective test data, while it tries to make the process entertaining for the developers. Rojas et al. [32] have introduced Code Defenders which is de-

signed based on the gamification approach. This tool is designed to conduct effective mutation testing. To this end, a player is either an attacker that tries to generate mutants or a defender that attempts to generate effective test cases to kill the mutants. García et al. [51] introduced a novel comprehensive framework by which developers are enabled to establish gamified applications for software engineering. The framework comprises an ontology, a methodology, and a gamification engine. To understand the efficacy of gamification for teaching purposes, de Jesus et al. [52] conducted a systematic mapping study to investigate the application of gamification in the context of software testing. Then, they developed a gamified approach by which the researchers established training courses on functional testing. A major observation of the study was that the participants found the gamified training attractive and funny.

2.3 Summary

In this section, an elaboration on the concepts that are necessary to grasp the contribution of this paper was presented. Particularly, the preliminaries of software testing and fault localization techniques were elucidated. Then, computer games as well as gamification were illuminated. Gamification is the application of games in serious practical and industrial tasks. Next, the studies that are related to this paper were reviewed. From debugging assistance category of techniques, it can be learnt that they have proven useful tools for developers. However, debugging is in nature, an exploration task. As such, it is a time-consuming and arduous task and needs the developer to remain patient and enthusiastic. From the fault localization category of techniques, it can be learnt that the current AFL techniques, particularly SBFL ones, have shown promising experimental effectiveness. However, they do not currently generalize to every situation and human intervention becomes mandatory. Finally, from the gamification category of techniques, it can be learnt that gamified software testing applications can introduce enthusiasm to the users and can help them keep attracted to the assigned task.

Based on the lessons learned and the successful results of gamified software testing applications, the authors are motivated to apply gamification in post-test tasks of the software debugging process. In particular, this paper is aimed to introduce a gamified approach for fault localization and program repair. In the next section, the details of fulfilling this objective are articulated.



3 The Proposed Approach

In this section, a new approach, namely *Gamifault*, is introduced that is intended to be used as a debugging assistance tool. The difference between Gamifault with the existing tools lies in the incorporation of game elements and mechanics in the tool functionality. These augmented functionalities are adapted to the context of program debugging to act as stimuli and provide motivation and enthusiasm to the process. That is, it sought to develop a gamified tool, which tries to mitigate the issues with the longsome and boring nature of the debugging process. The ultimate goal is to help developers fix more faults and release higher-quality software systems.

As can be seen in Figure 1, Gamifault receives as input, the source code of a faulty program p and an associated test suite TS for p . In general, any AFL technique can be incorporated into Gamifault. Hereafter the applied AFL technique is called *afl*. The test suite must involve at least a failing test case that acts as a fault oracle. Then, Gamifault proceeds with three major steps as per the following.

Step 1: Automatic Fault Localization. The incorporated fault localization technique *afl* is applied on p based on TS . The result of this step is a ranked (arranged) list of code elements within p according to the fault-proneness estimation of code elements. In particular, a calculation is carried out on p to assign the code elements some sort of score. Based on the scores, the code elements within p are ranked. A top-ranked code element c indicates that it is highly likely that c is responsible for the fault in p . For instance, if the applied *afl* is an SBFL technique and the desired code element is chosen at the statement level, the calculation comprises instrumentation on p followed by computing the suspiciousness scores for every statement. This step helps significantly narrow down the search space of faulty code elements. As a consequence, the required time for the developer (player) to pinpoint the faulty code elements, is expected to be substantially reduced [10].

Step 2: Code Modification. In this step, the developer leverages the information from the last step and tries to identify the exact code elements in p that are responsible for the fault. Then, the developer modifies p in an attempt to fix the fault.

Step 3: Context Gamification. In this step, the modified program and TS are used to achieve some information from the last activity of the developer on p . Then, several game mechanics and elements are applied to this information and the results are presented to the developer. The developer is allowed to accomplish the debugging tasks in a pre-determined

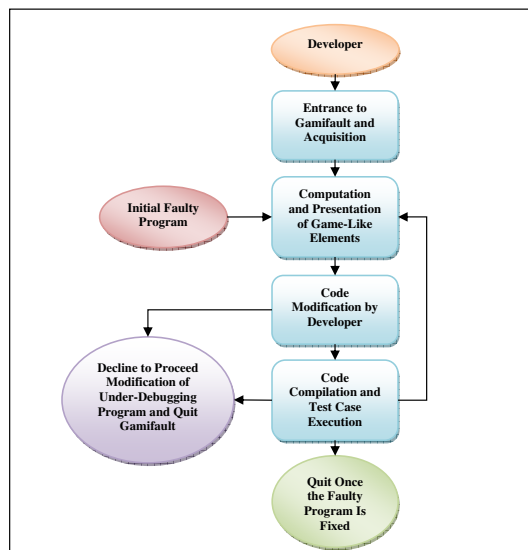


Figure 2. The Flowchart Demonstration of Gamifault Gamification.

time slot. In addition, the time-lapse is shown to the developer, which acts as a game mechanic in the Gamifault approach. Put simply, the developer observes the progress of debugging as a gamified context, which is proportionate to his/her performance. This gamified context is intended to retain the attention of the developer and makes the process motivational and enthusiastic.

Gamifault considers the modified program at Step 3 as a new program p and goes back to Step 1. The process is repeated until the modified program is passed against all test cases within TS or a pre-determined timeout is exhausted. After the fault-localization process, the third step involves code modification, which is also a manual task of the developer.

3.1 Gamification of Debugging Context

In this section, the details of enforcing game mechanics and elements in the process of debugging by Gamifault are articulated. Particularly, in this step, Gamifault reacts to the developer in a gamified manner. That is, it provides some feedback to the developer based on his/her modifications to the faulty program. The feedbacks offer two functionalities. For one, they inform the developer how close the modified faulty program is to the correct one. For two, they are intended to stimulate the developer to proceed.

The gamification dynamics that Gamifault is meant to foster are *acquisition*, *retention*, and *involvement*. The acquisition part concerns itself with the act of entering the developers into the Gamifault system. The retention part is associated with incentivizing the developers to come back to the Gamifault system



and keep using it. In the involvement part, Gamifault attempts to incorporate and present game-like effects while the developer is involved with modifying and fixing the faulty code.

The flowchart in Figure 2 presents how Gamifault helps developers more effectively debug the program. In non-gamified situations, typically, a developer considers modifying the faulty code irrespective of his/her previous actions. The developer may be ignorant of how much time has elapsed and how close is the current faulty program to the correct version. However, in the gamified context, which is the case for Gamifault, the developer is notified with the elapsed time, the remaining time, and the amount of proximity to the correct, fixed version of the given faulty program, in terms of the number and the names of the passing test cases and the remaining failing ones. These data can inform the developer of his/her progress. In addition, obtained point of the developer, along with his/her rank and level, is shown, which can act as persuasive motives to proceed. By the time the developer stops modifying the code and re-runs the test cases, the game-like data are updated.

3.1.1 Acquisition

To realize the acquisition part, two approaches are employed, which are advertising Gamifault and introducing it through the Gamifault users. To advertise Gamifault, one can use social networks such as the channels and groups within Telegram and WhatsApp as well as Instagram posts. In addition, word-of-mouth marketing can be used.

To introduce Gamifault to new users, several game elements are used to act as a referral engine. The elements are intended to incentivize the developers to inform and ask their friends, families, and acquaintances to use Gamifault. The major element that is applied for this purpose is an achievement. For each user invitation, some achievements are considered that ultimately serve Gamifault as a way for promotion and proliferation of the users. The sub-elements of achievement include point, badge, scheduled score, activity feed, and status.

To realize the point element, Gamifault provides a way to invite new users by a current user. Then, the current user achieves an extra point per new user invitation. The badge element is applied as a scoring mechanism to encourage the users to keep utilizing the Gamifault to complete their tasks. Gamifault offers several badges to a current user who invites new users. The scheduled score is a game element that attempts to establish some constraints for the user to increase the efficiency of his/her activities. To this end, Gamifault allows inviting at most two new users

per day. This design decision is considered to enforce a current user to invite more specialized and skilled users. With the activity feed, it is assumed that presenting the status of each user may foster the motivation of other users. To this end, Gamifault applies a dashboard to demonstrate the obtained scores of users. The functionality of the status element is to offer some privileges to the users with higher scores. Gamifault allows privileged users to choose harder faulty programs, which can indicate higher skill and expertise of the user.

3.1.2 Retention

To incentivize users to permanently recur Gamifault and keep applying it for debugging, it is needed to properly identify the key activities and evaluate them. Several game elements are used to fulfill the mentioned objective. They include achievement, bonus, level, progression, leaderboard, onboarding, cascading info theory, scheduled score, countdown, ownership, loss aversion, user profile, avatar, competition, endless, discovery, activity feed, and notifier.

The first retention element is an achievement, which is employed to motivate the players. Gamifault utilizes two kinds of achievements, namely, point and badge. Once a player achieves a certain objective, some points can be assigned to further motivate the player to keep using the system. In Gamifault, badges are shown based on the number of debugged programs and the number of times that a player signs in to the system and uses Gamifault to debug a program.

Regarding the bonus, Gamifault considers offering a bonus to a user who can complete the assigned mission in the determined time interval. In addition, a bonus is given to a user that completes the questionnaire or his/her profile. Gamifault presents the most recent level in which a user is playing. However, Gamifault does not enforce any limitation to the number of levels that can be followed by a user. Nevertheless, some extra points are assigned to the user who reaches higher levels.

Progression is another game element by which Gamifault attempts to motivate users to keep playing. Offering a badge is an instance of user progression. In addition, the movement to the next level, leaderboard, and progression bar are other means that serve Gamifault as the enforced game elements.

Another game element that is employed by Gamifault is a leaderboard. With this element, Gamifault is intended to present the highest scores to make a further incentive for the players. To realize the leaderboard Gamifault considers showing the name, point, rank, and avatar of the 10 players with the highest



total points. In addition, an onboarding game element is provided in Gamifault to serve the player as the user manual of the system and individual levels.

The objective of cascading info theory is to present the information of the project in small fragments to the user. With a scheduled score, a user can obtain some scores after a certain milestone is achieved. Gamifault offers scores in terms of points that are given upon completion of a certain number of projects. The countdown game element is intended to put constraints on the allowed time to complete a project. The time constraints can be determined based on the difficulty of the assigned programs. A user is expected to focus on the assigned project so that he/she can complete the project before the time is exhausted.

The ownership mechanism intends to persuade the user to consider the assigned project as his/her property. In this way, the user may attempt to achieve the best results. Gamifault offers five credits to a user upon the first usage of the system. The given credits can also act as a loss-aversion mechanism for a user. In case a user fails to repair the program, the user's credit is decremented by 0.5. alternatively, if the user withdraws the task or the allowed time is exhausted, the user's credit is decremented by 1. Otherwise, in the case that the uses can successfully repair the given faulty program, his/her credit is incremented by 1.

Gamifault offers a profile section in which a user's personal information can be provided. For Gamifault, the information regarding the debugged programs is also recorded in the user's profile. The avatar is a game element by which the characteristics of a user are demonstrated. Gamifault allows a user to set an avatar in his/her profile to make some sort of anthropomorphic feelings. In addition, Gamifault offers a competition game element that is intended to motivate users to spend more effort during the assigned task. The competition allows a user to be informed of the status of other users, and consequently, the user may try to be more engaged to outperform other users.

The endless game element aims to infuse the user that further game levels remain. As a result, the user can be motivated to keep using the system or recur the system more frequently. The discovery game element is a means to incentivize the user to explore the system to have an experience from every aspect of the system. For example, Gamifault assigns a deadline to complete a certain task and moves forward to the next levels of the system. With the activity feed element, Gamifault seeks to inform players of the progress of the current task at hand. Gamifault makes use of a bulletin board, in which the most recent events are shown. In addition, a notifier is used by which Gamifault informs the player upon promoting to a higher level.

3.1.3 Involvement

To involve the active participation of the developer during the debugging, Gamifault offers scores for various situations. For one thing, Gamifault assigns certain scores according to the difficulty of the undertaken task. For a player with credit $(m/5)$, Gamifault offers a score of $(n * 15) * (m/5)$ if the player could successfully debug a program with difficulty factor n . For another thing, in case a player has been using Gamifault for k consecutive days, a score of $(k * 2.5)$ is added to the player's score. Table 2 lists the scores that Gamifault offers based on the player's various activities.

4 Evaluation

To evaluate Gamifault, implemented a web-based prototype tool¹ using ASP.NET and Python languages was implemented. Figure 3 depicts an execution snapshot of the prototype tool.

In the tool, Tarantula was used as an SBFL technique embedded in Gamifault. In addition, Cobertura² was used to serve us as a Java code coverage tool. As subject programs, 46 faulty programs were randomly chosen from Code4Bench [22], in which high-quality test cases, along with associated test oracles, are provided. The subject programs are written in Java language by international programmers from all over the world. The programs implement various algorithmic problems. As the compiler of Java programs, Java Development Kit (JDK) 8 was used. In addition, a graphical environment was prepared in our tool that visually highlights the code. In particular, a code highlighted in red color indicates that it has been executed by a failing test case and implies a dangerous situation from the gamification viewpoint. A code highlighted in green is intended to convey that it has been executed by a passing test case, yielding a safe situation. A code highlighted in yellow specifies a caution situation. That is, the code has been executed by both passing and failing test cases. In case the whole suite of test cases is passed against the given program, further SBFL computations are not required at all. The mentioned scheme of code highlighting from Jones et al. [53] was adopted in our tool. In Figure 4, it can be seen shown how a participant can invite friends, thereby promoting Gamifault and increasing his/her points. The main mechanism is by submitting an email to a friend.

To demonstrate the effectiveness of Gamifault, a couple of groups of participants were considered where

¹ <https://github.com/akhalilian/Gamifault>

² <https://cobertura.github.io/cobertura/>



Table 2. The Scoring Scheme Used by Gamifault Based on the Player'S Activities.

Activity No.	Description	Score
1	Filling optional information fields in the player's profile	7.5
	Filling necessary information fields in the player's profile including the user's email	20.0
2	Correct debugging of the faulty code by the user	15.0
3	For each time that the user logs in to the Gamifault	2.5
4	Consecutive application of Gamifault	5.0
5	Announcing status on social networks	10.0
6	Filling questionnaire inquiries	7.5
7	Consecutive debugging	17.5
8	The current player invites a new user to the Gamifault	4.1
9	The current player invites a new user to the Gamifault and the invited user signs in to the Gamifault	8.2
10	The current player invites a new user to the Gamifault, the invited user signs in and starts debugging	12.5
11	The received score at level l ; starts at level number one and is increased by one.	$(\frac{l}{0.25})^2$

The screenshot shows the 'FAULT LOCALIZATION' interface of the Gamifault Prototype Tool. At the top, there are navigation tabs: 'Code Selection', 'Fault Localization', 'Leaderboard', 'Setting', and 'Exit'. The main area is divided into three sections:

- Code Editor:** Displays Java code for a game problem. The code includes a class 'Game' with a static method 'g' that takes an integer 'n' and a string 's' as input. It counts the number of '0's and '1's in the string and returns the minimum number of moves required to win. The code is highlighted with alternating green and orange lines.
- Test Case List:** A vertical list on the right side shows three test cases: 'GameTest1', 'GameTest2', and 'GameTest3'. 'GameTest1' is currently selected and highlighted in light blue.
- Remaining Time:** A counter at the top right shows '284/712'.

At the bottom of the interface, there is a status bar that says 'no compile error'.

Figure 3. An Execution Snapshot of the Gamifault Prototype Tool.

Table 3. The questions and associated choices used in the survey experiment. “A”, “R”, “C”, and “S” correspond to attention, relevance, confidence, and satisfaction, respectively.

Number	Questions and Choices	Model Component
1	I can proceed with software debugging with no exhaustion for ... a) very short time. b) short time. c) long time. d) very long time.	A
2	I have expected to solve (debug)... several programs. a) few b) a few c) a multitude d) a vast	C
3	The amount of time that I have spent debugging is. . . a) much less than what I have considered. b) less than what I have considered. c) more than what I have considered. d) much more than what I have considered.	C
4	If I face an issue in the course of debugging, I. . . a) decline to debug. b) usually try to resolve it. c) decline to debug quickly. d) decline to debug much quickly.	R
5	When I debug collectively, ... a) I am very little interested in completing before others. b) I am little interested in completing before others. c) I am much interested in completing before others. d) I am very much interested in completing before others.	C
6	I am not noticed as time elapses. . . a) I do not focus on debugging at all I miss the time elapse. b) I do not deeply focus on debugging and I miss the time elapse. c) I am focused on debugging. d) I am highly focused on debugging.	A
7	I ... debug the code because it is ... exhilarating for me. a) do not continuously, much little b) do not continuously, little c) continuously, much d) continuously, very much	S
8	I ... recommend Gamifault to my friends after every debug. a) seldom b) weakly c) highly d) strongly	S
9	Whenever I have some time, I keep on debugging. a) seldom b) weakly c) highly d) strongly	R
10	When I face a multiple-fault program, ... a) I am seldom motivated to resolve all faults. b) I am weakly motivated to resolve all faults. c) I am highly motivated to resolve all faults. d) I am strongly motivated to resolve all faults.	R

one group employs Gamifault, which is the gamified debugging environment, and another group works with the non-gamified version, in which only SBFL scores are presented. Gamifault and its non-gamified version serve us as tools for test and control experiments, respectively. The non-gamified version is needed that allows us to assess whether the absence of game-like elements might result in performance degradation of the debugging process, hence gamified debugging would

be a promising approach. The non-gamified version is not meant solely to compute code coverage. Rather, it automatically computes the suspiciousness scores of each statement based on the Tarantula technique. The aim is to provide a debugging environment analogous to Gamifault, in which just game-like elements are omitted.

The participants have been selected from the bach-



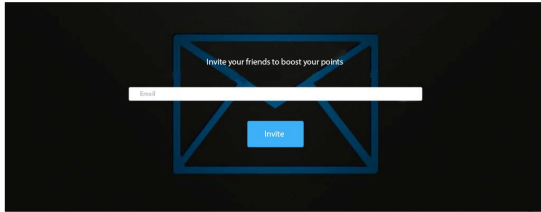


Figure 4. Invitation of Friends to Promote Gamifault.

elor students of computer engineering at Shahid Beheshti University³, which is one of the top universities in Tehran, the capital city of Iran. Each participant is required to be at least a sophomore student and is familiar with the Java language. As a consequence, two groups of participants were collected, with each group comprising 15 persons. The ages of participants range from 19 to 23 years old with 22.5 years on average. The participants were allowed to use the system at an arbitrary period of the determined time interval, which was set to seven days. In addition, a participant can leave the system at any desired time. Figure 5 depicts some screenshots of running Gamifault.

4.1 Results

In the gamified version of the system, that is the Gamifault itself, 5 participants did not log in to the system at all, whereas 10 remaining participants continuously kept using the system during the determined time interval. 10 users of the gamified version have spent 982 minutes (16.36 hours) with Gamifault and could have successfully debugged 95 faulty programs. On average, each participant has spent 65.4 minutes on debugging, which includes fault localization and program repair. The average debugging time for male and female participants has been obtained as 82.5 and 46 minutes, respectively. For 10 participants who took part in the experiment with Gamifault, a range of 27 seconds to 18:06 minutes was taken to reach the sixth level of Gamifault.

In the non-gamified version of the system, 9 participants declined to use the system and did not use it at all. The users of this group have spent 254 minutes and could have successfully debugged 7 faulty programs. The average debugging times for the users of the non-gamified system are 9.65 and 23.35 minutes, respectively. In addition, the users have spent a minimum of 28 seconds and a maximum of 20:23 minutes using the system whereas the users could achieve at most the fourth level. On average, the participants spent 16.9 minutes using the system.

The analysis of the results shows that with the gamified version of the system, the faulty programs have

been debugged in less time, as compared to the non-gamified version. In addition, with Gamifault, the number of debugged programs has dramatically increased by a factor of nearly 13. This result can be attributed to the participants' eagerness and enthusiasm established by the game elements embedded in Gamifault.

To evaluate the effectiveness of Gamifault, the measures throughput, average lifetime play, and expected contribution were measured, which were proposed by Von Ahn and Dabish [54]. Throughput is defined as the average number of programs that are debugged per one participant/hour. Average lifetime play is defined as the average time a participant spends using the systems for other participants. The expected contribution is obtained from the multiplication of throughput and the average lifetime play.

As a result of applying the measures to our experimental data with Gamifault, the throughput would be 5.8, which is 95 debugged programs divided by 16.36 hours. The average lifetime play is obtained from a division of 982 by 15, yielding 65.46 minutes or 1.09 hours. From the last two measures, it turns out that the expected contribution would be 1.09 multiplied by 5.8, which gives 6.32. Applying the measures to the experimental data with the non-gamified system, 1.65 programs per participant/hour, 17.33 minutes (0.28 hours), and 1.18 were obtained for the throughput, average lifetime play, and expected contribution, respectively.

Overall, it can be concluded that Gamifault can noticeably outperform traditional debugging systems. Gamifault seems to introduce a large amount of motivation and exhilaration to the developers and make them keep debugging the programs at hand.

4.2 The Survey

The core proposals of this research are the introduction of crowdsourcing into the debugging task while motivating the participants. Gamifault is designed to offer these properties. The ultimate goal is to improve the effectiveness of human activities in debugging faulty software. Accordingly, it is sought to assess the satisfaction of the participants with Gamifault. To this end, a questionnaire-based survey was prepared in which 10 questions were asked from the participants who have used the gamified and non-gamified systems, namely test and control groups, respectively. The questions are designed based on the ARCS motivational model [55]. The aim is to assess the overall experience and motivation of the participants when applying both gamified and non-gamified versions. The model comprises four components, namely, attention (A), relevance (R), confidence (C), and satisfaction

³ <https://en.sbu.ac.ir/>



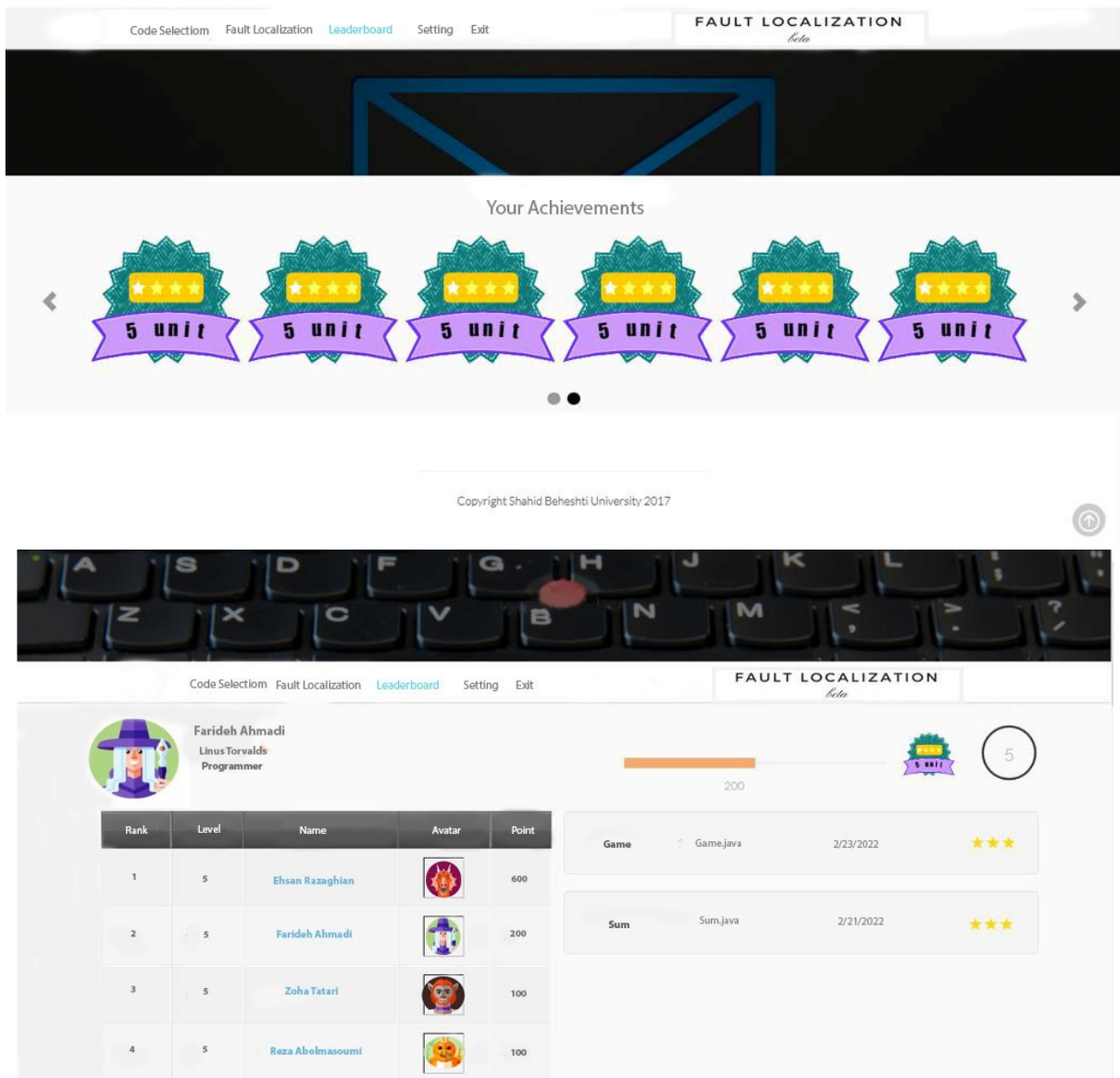


Figure 5. Snapshots of Gamifault Execution.

(S). The A component is concerned to the amount to which the participant is interested in the content and problem-solving. The R component deals with how much the participant is accustomed to the assigned activity. The C component points to the fact that the confidence of the participant in his/her ability to accomplish the task would lead to extra motivation and a tendency to proceed with the activity. Finally, the S component indicates that the participant needs to have a sense of value in the activity to justify spending unremitting attempts to complete it. The questions are presented in Table 3.

In the survey experiment, the amount of motivation that Gamifault introduces to the participants is considered the dependent variable. The survey was con-

ducted in both pre-experiment and post-experiment manners. The SPSS version 18 was used to analyze the results. Table 3 demonstrates the mean and standard deviation of pre-experiment and post-experiment answers for both test and control groups. As can be seen in Table 4, the mean and standard deviation of the motivation for the test group has increased, as compared to the control group.

To compute the statistics presented in Table 4, we have used the Likert-Scale⁴ method [56, 57]. In survey studies, in which questionnaires are used, the Likert-Scale method is widely used to transform qualitative answers to quantitative measures. Then, the scores

⁴ https://en.wikipedia.org/wiki/Likert_scale



Table 4. The mean and standard deviation of the answers for test and control groups, measured in pre-experiment and post-experiment manners.

Group	Mean	Standard Deviation	Number
Control	21.8000	3.91335	15
Test	30.0667	4.43234	15
Total	25.9333	5.54563	30

associated with the answers of each questionnaire are summed to achieve the total score of that questionnaire, that is, the score of a participant's survey. In Table 4, the mean and standard deviation statistics are obtained from the scores of our survey.

4.3 Discussion and Threats to Validity

Gamifault is a gamified debugging environment that can be used to resolve faults. It is meant to make an attractive environment so that the developer proceeds debugging with higher motivation and enthusiasm. Observing from the code review domain, Gamifault highlights the suspicious statements. In search of faulty statement(s), a developer can start reviewing high-score, suspicious statements to lower ones. Gamifault does not provide any novelty from the gamification perspective. Rather, the novelty lies in the adaptation and application of gamification potentials into the laborious task of debugging. Regarding the quality of the repaired codes, our investigations showed that the repaired codes pass against every test case in the associated test suite and the codes are readable, but not necessarily optimized, clean code.

In the experiments, 30 users participated, which might be an impediment to the generalization of the results. However, the choice of 15 participants for each of the control and test groups is a statistically standard method in studies involving human intervention. Next, Gamifault is hardly related to the quality of the test suite. Hence, a low-quality, inadequate test suite that lacks at least a failing test case can render Gamifault a less effective approach. To mitigate this threat, the subject programs were taken from the Code4Bench suite of programs. The programs in Code4Bench are provided with high-quality test cases. In addition, Gamifault does not provide any facility to supply the deficiencies associated with unskillful or junior developers. As such, the level of expertise in programming among the users can either improve or exacerbate the overall efficacy of Gamifault. To mitigate this threat, the users were chosen from the bachelor students of computer engineering at Shahid Beheshti University, which is one of the top univer-

sities in Tehran. Furthermore, each chosen user was required to be at least a sophomore student as well as familiar with the Java language. As a result, a bare minimum of programming expertise was considered.

5 Conclusions

The objective of this paper was to present a gamified approach that can make the debugging of a faulty program attractive. This objective was accomplished by introducing Gamifault which can be used to motivate developers during the debugging process, that is fault localization and program repair. Gamifault leverages an existing spectrum-based fault localization technique to rank the code elements according to their suspiciousness score. The developer uses the suggested ranked code elements and attempts to find the statements that are responsible for the fault. Then, the developer modifies the program in an attempt to fix the fault. Gamifault uses a scoreboard to show the success rate of the developer, that is the number of passed test cases on the modified program. The process is repeated similarly until the developer hopefully can fix the faulty program. To evaluate Gamifault, a prototype tool was developed and used on 46 subject programs taken from Code4Bench. For the non-gamified version, we have obtained 1.65, 17.33, and 1.18 for throughput, average lifetime play, and expected contribution, respectively. for the gamified version, the experimental results were obtained as 5.8, 65.46, and 6.32 for throughput, average lifetime play, and expected contribution, respectively. in addition, using a questionnaire-based survey, we found that the motivation for the test group has increased, as compared to the control group. It turned out that using the Gamifault, most of the faulty programs can be fixed in less than two minutes on average.

In the future, this work can be extended in several ways. First, further implementations for other popular programming languages such as Python and Kotlin can be presented and evaluated. Second, additional experiments can be conducted on larger programs. Third, several alternative fault localization techniques can be used and investigated.

References

- [1] G. Catolino, F. Palomba, A. Zaidman, and F. Ferrucci. Not all bugs are the same: Understanding, characterizing, and classifying bug types. *Journal of Systems and Software*, 152:165–181, 2019. doi:10.1016/j.jss.2019.03.002.
- [2] G. Rodríguez-Pérez, G. Robles, A. Serebrenik, A. Zaidman, D. M. Germán, and J. M. Gonzalez-



- Barahona. How bugs are born: a model to identify how bugs are introduced in software components. *Empirical Software Engineering*, 25:1294–1340, 2018. doi:10.1007/s10664-019-09781-y.
- [3] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, P. McMinn, and A. Bertolino. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*, 86(8):1978–2001, 2013. doi:10.1016/j.jss.2013.02.061.
- [4] J. Xuan, M. Martinez, F. Demarco, M. Clement, S. L. Marcote, T. Durieux, D. Le Berre, and M. Monperrus. Nopol: Automatic repair of conditional statement bugs in java programs. *IEEE Transactions on Software Engineering*, 43(1):34–55, 2016. ISSN 0098-5589. doi:10.1109/TSE.2016.2560811.
- [5] M. Zhang, Y. Li, X. Li, L. Chen, Y. Zhang, L. Zhang, and S. Khurshid. An Empirical Study of Boosting Spectrum-Based Fault Localization via PageRank. *IEEE Transactions on Software Engineering*, 47(6):1089–1113, 2021. ISSN 0098-5589. doi:10.1109/TSE.2019.2911283.
- [6] M. Beller, N. Spruit, D. Spinellis, and A. Zaidman. On the dichotomy of debugging behavior among programmers. In *Proceedings of the 40th International Conference on Software Engineering*, pages 572–583. ACM, 2018. doi:10.1145/3180155.3180175.
- [7] S. Kaleeswaran, V. Tulsian, A. Kanade, and A. Orso. MintHint: automated synthesis of repair hints. In *Proceedings of the 36th International Conference on Software Engineering*, page 266–276. ACM, 2014. doi:10.1145/2568225.2568258.
- [8] P. Ma, H. Cheng, J. Zhang, and J. Xuan. Can this fault be detected: A study on fault detection via automated test generation. *Journal of Systems and Software*, 170:110769, 2020. doi:10.1016/j.jss.2020.110769.
- [9] P. S. Leitao-Junior, D. M. Freitas, S. R. Vergilio, C. G. Camilo-Junior, and R. Harrison. Search-based fault localisation: A systematic mapping study. *Information and Software Technology*, 123:106295, 2020. doi:10.1016/j.infsof.2020.106295.
- [10] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa. A Survey on Software Fault Localization. *IEEE Transactions on Software Engineering*, 42(8):707–740, 2016. ISSN 0098-5589. doi:10.1109/TSE.2016.2521368.
- [11] L. Gazzola, D. Micucci, and L. Mariani. Automatic software repair: a survey. *IEEE Transactions on Software Engineering*, 45(1):34–67, 2019. ISSN 0098-5589. doi:10.1109/TSE.2017.2755013.
- [12] K. Liu, L. Li, A. Koyuncu, D. Kim, Z. Liu, J. Klein, and T. F. Bissyandé. A critical review on the evaluation of automated program repair systems. *Journal of Systems and Software*, 171:110817, 2021. doi:10.1016/j.jss.2020.110817.
- [13] R. Van Rozen. Languages of Games and Play: A Systematic Mapping Study. *ACM Computing Surveys (CSUR)*, 53(6):1–37, 2021. doi:10.1145/3412843.
- [14] D. de Paula Porto, G. M. de Jesus, F. C. Ferrari, and S. C. P. F. Fabbri. Initiatives and challenges of using gamification in software engineering: A Systematic Mapping. *Journal of Systems and Software*, 173:110870, 2021. doi:10.1016/j.jss.2020.110870.
- [15] B. Morschheuser, L. Hassan, K. Werder, and J. Hamari. How to design gamification? A method for engineering gamified software. *Information and Software Technology*, 95:219–237, 2018. doi:10.1016/j.infsof.2017.10.015.
- [16] S. Stieglitz, C. Lattemann, S. Robra-Bissantz, R. Zarnekow, and T. Brockmann. *Gamification Using Game Elements in Serious Contexts*. Berlin: Springer, 2017.
- [17] A. Khalilian, A. Baraani-Dastjerdi, and B. Zamani. CGenProg: Adaptation of cartesian genetic programming with migration and opposite guesses for automatic repair of software regression faults. *Expert Systems with Applications*, 169:114503, 2021. doi:10.1016/j.eswa.2020.114503.
- [18] M. Monperrus. Automatic Software Repair: A Bibliography. *ACM Computing Surveys (CSUR)*, 51(1):1–24, 2019. doi:10.1145/3105906.
- [19] A. Zakari, S. P. Lee, R. Abreu, B. H. Ahmed, and R. A. Rasheed. Multiple fault localization of software programs: A systematic literature review. *Information and Software Technology*, 124:106312, 2020. doi:10.1016/j.infsof.2020.106312.
- [20] D. Zou, J. Liang, Y. Xiong, M. D. Ernst, and L. Zhang. An Empirical Study of Fault Localization Families and Their Combinations. *IEEE Transactions on Software Engineering*, 47(2):332–347, 2019. ISSN 0098-5589. doi:10.1109/TSE.2019.2892102.
- [21] J. A. Jones and M. J. Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, page 273–282. ACM, 2005. doi:10.1145/1101908.1101949.
- [22] A. Majd, M. Vahidi-Asl, A. Khalilian, A. Baraani-Dastjerdi, and B. Zamani. Code4Bench: A multidimensional benchmark of Codeforces data for different program analysis techniques. *Journal of Computer Languages*, 53:38–52, 2019. doi:10.1016/j.col.2019.03.006.
- [23] P. Ammann and J. Offutt. *Introduction to soft-*



- ware testing. Cambridge University Press, 2016.
- [24] R. Abreu, P. Zoetewij, and A. J. Van Gemund. On the Accuracy of Spectrum-based Fault Localization. In *Testing: Academic and industrial conference practice and research techniques-MUTATION (TAICPART-MUTATION 2007)*, pages 89–98. IEEE, 2007. ISBN 0-7695-2984-4. doi:10.1109/TAIC.PART.2007.13.
- [25] X. Xie, T. Y. Chen, F. Kuo, and B. Xu. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. *ACM Transactions on software engineering and methodology (TOSEM)*, 22(4):1–40, 2013. doi:10.1145/2522920.2522924.
- [26] Y. Xiaobo, L. Bin, and W. Shihai. A Test Restoration Method based on Genetic Algorithm for effective fault localization in multiple-fault programs. *Journal of Systems and Software*, 172:110861, 2021. doi:10.1016/j.jss.2020.110861.
- [27] F. Y. Assiri and J. M. Bieman. Fault localization for automated program repair: effectiveness, performance, repair correctness. *Software Quality Journal*, 25:171–199, 2017. doi:10.1007/s11219-016-9312-z.
- [28] S. Amiri-Chimeh, H. Haghighi, M. Vahidi-Asl, K. Setayesh-Ghajar, and F. Gholami-Ghavamabad. Rings: A Game with a Purpose for Test Data Generation. *Interacting with Computers*, 30(1):1 – 30, 2018. ISSN 1873-7951. doi:10.1093/iwc/iww043.
- [29] J. McGonigal and R. Broken. *Why Games Make Us Better and How They Can Change the World*. 2011. US.: Penguin Group, 2011.
- [30] S. Cooper, A. Treuille, J. Barbero, A. Leaver-Fay, K. Tuite, F. Khatib, A. C. Snyder, M. Beenen, D. Salesin, and D. Baker. The challenge of designing scientific discovery games. In *Proceedings of the Fifth international Conference on the Foundations of Digital Games*, page 40–47. ACM, 2010. doi:10.1145/1822348.1822354.
- [31] A. De la Croix and J. Skelton. The simulation game: an analysis of interactions between students and simulated patients. *Medical Education*, 47(1):49–58, 2013.
- [32] C. Parnin and A. Orso. Code Defenders: Crowdsourcing Effective Tests and Subtle Mutants with a Mutation Testing Game. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 677–688. IEEE, 2017. ISBN 978-1-5386-3868-2. doi:10.1145/2001420.2001445.
- [33] A. Darejeh and S. S. Salim. Gamification Solutions to Enhance Software User Engagement—A Systematic Review. *International Journal of Human-Computer Interaction*, 32(8):613–642, 2016. doi:10.1080/10447318.2016.1183330.
- [34] K. Seaborn and D. I. Fels. Gamification in theory and action: A survey. *International Journal of Human-Computer Studies*, 74:14–31, 2015. doi:10.1016/j.ijhcs.2014.09.006.
- [35] Q. Wu, Y. Zhu, and Z. Luo. A gamification approach to getting students engaged in academic study. *Bulletin of the IEEE Technical Committee on Learning Technology*, 17(4):26–29, 2015.
- [36] M. Brodie, S. Ma, G. Lohman, L. Mignet, N. Modani, M. Wilding, J. Champlin, and P. Sohn. Quickly Finding Known Software Problems via Automated Symptom Matching. In *Second International Conference on Autonomic Computing (ICAC'05)*, pages 101–110. IEEE, 2005. ISBN 0-7695-2276-9. doi:10.1109/ICAC.2005.49.
- [37] R. Abraham and M. Erwig. Goal-directed debugging of spreadsheets. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, pages 37–44. IEEE, 2005. ISBN 0-7695-2443-5. doi:10.1109/VLHCC.2005.42.
- [38] W. Weimer. Patches as better bug reports. In *Proceedings of the 5th international conference on Generative programming and component engineering*, page 181–190. ACM, 2006. doi:10.1145/1173706.1173734.
- [39] B. Ashok, J. Joy, H. Liang, S. K. Rajamani, G. Srinivasa, and V. Vangala. DebugAdvisor: A recommender system for debugging. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 373–382. ACM, 2009. doi:10.1145/1595696.1595766.
- [40] D. Jeffrey, M. Feng, N. Gupta, and R. Gupta. BugFix: A learning-based tool to assist developers in fixing bugs. In *2009 IEEE 17th International Conference on Program Comprehension*, pages 70–79. IEEE, 2009. ISBN 978-1-4244-3998-0. doi:10.1109/ICPC.2009.5090029.
- [41] B. Hartmann, D. MacDougall, J. Brandt, and S. R. Klemmer. What would other programmers do: suggesting solutions to error messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1019–1028. ACM, 2010. doi:10.1145/1753326.1753478.
- [42] T. Barik, Y. Song, B. Johnson, and E. Murphy-Hill. From quick fixes to slow fixes: Reimagining static analysis resolutions to enable design space exploration. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 211–221. IEEE, 2016.
- [43] X. Mao, Y. Lei, Z. Dai, Y. Qi, and C. Wang. Slice-based statistical fault localization. *Journal of Systems and Software*, 89:51–62, 2014. doi:10.1016/j.jss.2013.08.031.
- [44] A. Nath and P. Domingos. Learning tractable



- probabilistic models for fault localization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016. doi:10.1609/aaai.v30i1.10175.
- [45] C. Liu, L. Fei, X. Yan, J. Han, and S. P. Midkiff. Statistical Debugging: A Hypothesis Testing-Based Approach. *IEEE Transactions on Software Engineering*, 32(10):831–848, 2006. doi:10.1109/TSE.2006.105.
- [46] J. Tu, X. Xie, T. Y. Chen, and B. Xu. On the analysis of spectrum based fault localization using hitting sets. *Journal of Systems and Software*, 174:106–123, 2019. doi:10.1016/j.jss.2018.10.013.
- [47] C. Le Goues, T. Nguyen, S. Forrest, and W. Weimer. GenProg: A Generic Method for Automatic Software Repair. *IEEE Transactions on Software Engineering*, 38(1):54–72, 2012. doi:10.1109/TSE.2011.104.
- [48] C. Parnin and A. Orso. Are automated debugging techniques actually helping programmers? In *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, pages 199–209, 2011. doi:10.1145/2001420.2001445.
- [49] S. Arai, K. Sakamoto, H. Washizaki, and Y. Fukazawa. A Gamified Tool for Motivating Developers to Remove Warnings of Bug Pattern Tools. In *2014 6th International Workshop on Empirical Software Engineering in Practice*, pages 37–42. IEEE, 2014. ISBN 978-1-4799-6666-0. doi:10.1109/IWESEP.2014.17.
- [50] N. Tillmann, J. Bishop, N. Horspool, D. Perelman, and T. Xie. Code hunt: searching for secret code for fun. In *Proceedings of the 7th International Workshop on Search-Based Software Testing*, page 23–26. ACM, 2014. ISBN 978-1-5386-3868-2. doi:10.1145/2593833.2593838.
- [51] F. Garcia, O. Pedreira, M. Piattini, A. Cerdeira-Pena, and M. Penabad. A framework for gamification in software engineering. *Journal of Systems and Software*, 132:21–40, 2017. doi:10.1016/j.jss.2017.06.021.
- [52] G. M. de Jesus, F. C. Ferrari, L. N. Paschoal, Simone d. R. S. de Souza, de D. Paula Porto, and V. H. S. Durelli. Is It Worth Using Gamification on Software Testing Education? An Extended Experience Report in the Context of Undergraduate Students. *Journal of Software Engineering Research and Development*, 8:6–1, 2020. doi:10.5753/jserd.2020.738.
- [53] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *Proceedings of the 24th International Conference on Software Engineering*, page 467–477. ACM, 2002. doi:10.1145/581339.581397.
- [54] L. Von Ahn and L. Dabbish. Designing games with a purpose. *Communications of the ACM*, 51(8):58–67, 2008. doi:10.1145/1378704.1378719.
- [55] J. M. Keller. Motivational design of instruction. *Instructional design theories and models: An overview of their current status*, 1(1983):383–434, 1983.
- [56] K. A. Batterton and K. N. Hale. The Likert Scale What It Is and How To Use It. *Phalanx*, 50(2):32–39, 2017.
- [57] A. T. Jebb, V. Ng, and L. Tay. A review of key Likert scale development advances: 1995–2019. *Frontiers in psychology*, 12:637547, 2021. ISSN 0098-5589.



Shahrzad Sadat Mousavi Esfahani received her M.Sc. in Software Engineering from Shahid Beheshti university. Her research interests include software testing, fault localization, defect prediction, program repair, and gamification.



Mojtaba Vahidi-Asl received his B.Sc. from Amirkabir University of Technology, and his M.Sc. and Ph.D. from Iran University of Science and Technology. He is currently an assistant professor of software engineering at Shahid Beheshti University. His research interests include software testing, fault localization, defect prediction, program repair, and benchmarking.



Alireza Khalilian received his M.Sc. and Ph.D. in Software Engineering from the Iran University of Science and Technology and the University of Isfahan, respectively. His research interests include software regression testing, automatic program repair, and software security.



Parastoo Alikhani received her Ph.D. from Shahid Beheshti University in Information Technology in Higher Education. Her research interest includes e-learning, educational software design, qualitative research, and Extended Reality & Serious Games.



Bardia Abhari received his B.Sc. from Shahid Beheshti University. His research interests include software testing, fault localization, and software debugging.