Competitive Debugging

Toward Contests Promoting Debugging as a Skill

Patrick Rein
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
patrick.rein@hpi.uni-potsdam.de

Tom Beckmann
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
tom.beckmann@hpi.uni-potsdam.de

Leonard Geier
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
leonard.geier@hpi.uni-potsdam.de

Toni Mattis
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
toni.mattis@hpi.uni-potsdam.de

Robert Hirschfeld
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
robert.hirschfeld@uni-potsdam.de

Abstract

Debugging is an essential part of software development. Numerous tools and techniques to improve debugging have been proposed in research or developed in the industry. However, only a few of those see widespread use, and debugging only rarely is a primary teaching subject.

To promote debugging as a distinct skill, we propose *Competitive Debugging*, where participants compete on who can repair a failure the fastest or the most comprehensively. We further propose a format for debugging contests aimed at attracting and engaging participants to motivate them to improve their debugging skills. In our proposed format participants simultaneously work on the same failure or observe fellow participants during their debugging activity. To evaluate the format, we ran two pilots and one main event. We found that the format prompted participants to reflect on their debugging process, that the format allowed them to compare their debugging approaches to others through post-round discussions on their various approaches, and that the format was enjoyable and engaging for all participants. We present our format of a debugging contest, an evaluation of the trial runs we performed, and give guidance for other people who consider hosting a Competitive Debugging event.

Ultimately, we aim to provide developers with opportunities to improve their debugging skills. Our observations indicate that Competitive Debugging can provide such opportunities to train debugging techniques and learn new tools.

Onward! '22, December 8–10, 2022, Auckland, New Zealand
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9909-8/22/12.
https://doi.org/10.1145/3563835.3567665

CCS Concepts: • **Social and professional topics** → *Computing education*; • **Software and its engineering** → **Software maintenance tools**.

Keywords: debugging, competitive debugging, competition, contest, teaching, training

ACM Reference Format:
Patrick Rein, Tom Beckmann, Leonard Geier, Toni Mattis, and Robert Hirschfeld. 2022. Competitive Debugging: Toward Contests Promoting Debugging as a Skill. In *Proceedings of the 2022 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '22), December 8–10, 2022, Auckland, New Zealand.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3563835.3567665

1 Introduction

Programmers spend considerable time debugging the programs they are working on. Depending on the study, debugging takes up 13 % to 40 % percent of programmers' software development time [2, 23][28, p. 8-2]. Correspondingly, to improve debugging, the research communities on programming languages and software engineering have developed numerous debugging strategies and tools, such as generalized backward reasoning [33], time-travel debuggers [17, 24], and automatic debugging tools [32].

However, most programmers use few of these tools and techniques in practice, although they would benefit from them [8, 23]. Programmers either do not know about advanced tools [23] or do not know how to set them up and make efficient use of them [30]. The reason for this mismatch may be a lack of education and training on debugging [2, 22, 23].

At the same time, researchers also found that debugging skills can be taught [19]. Numerous studies have shown that when students learn and train debugging skills, they employ systematic debugging techniques more often and become more productive at debugging [19, 21]. Nevertheless, debugging seems to be seldom taught extensively and

Figure 1. The trophy for the first place of our first *Competitive Debugging* event: the golden "bug".

systematically. One reason for this lack of training may be that teachers and students do not regard debugging as much a distinct skill as they regard programming.

To make debugging more prominent as a skill, we propose the idea of *Competitive Debugging*. At a debugging contest, contestants from beginners to experts gather to competitively find and repair defects.

Along with highlighting debugging as a skill, these events also directly serve as a way to teach debugging. By observing fellow contestants, participants can learn how to best employ debugging tools and strategies or reflect on their own techniques through observing the debugging process of others.

Further, debugging contests may also help disseminate knowledge about the existence and application of advanced programming tools.

In this paper, we outline the basic idea of Competitive Debugging and fundamental considerations when setting up individual debugging contests. To illustrate how these may look like in practice, we describe three events from the perspective of organizers and participants: a pilot contest with researchers, a pilot contest for demonstrating advanced tools through Competitive Debugging, and an actual debugging contest with graduate-level students resulting in the awarding of the golden "bug" (see Figure 1).

In the following, we first discuss debugging skills and their training in some more detail, and discuss related computer science competition formats (see Section 2). We then describe the core idea of Competitive Debugging and what needs to be considered when planning a contest (see Section 3). Afterwards, we describe the setup, observations, and interview results from the two pilot contests (see Section 4) and the actual event (see Section 5)[1]. Finally, we discuss general considerations when planing a debugging contest (see Section 6).

2 Background

Debugging is a skill that requires knowledge and experience with debugging techniques and tools. At present, many programmers have received little education and training in debugging. Competitions can be a way to foster such education and training of a software development skill, as shown by a variety of competitions in other fields such as general programming or cybersecurity.

In the remainder of the paper, we will refer to the wrong section of code as the *defect* or *fault*, the wrong run-time state as the *infection*, and the observable, unexpected system behavior as the *failure*.

2.1 The Need for Debugging Education

Debugging is an integral activity during software development [28]. At the same time, debugging is a skill separate from programming, requiring specialized education and training. This is underlined by several studies that illustrate that, at least for students, good programmers are not necessarily good debuggers, while most good debuggers are also good programmers [1, 9, 20]. Although debugging is important in everyday software development, most software developers have not received considerable training on debugging techniques and tools [23].

Programmers developing software regularly debug their systems, to get code that they are currently working on running, to repair code written in the past, or to better understand the system and it actually behaves. Depending on the study, programmers do debugging between 13 % to 40 % of their programming time [2, 23][28, p. 8-2].

Although debugging takes up such a large portion of development time and debugging is a distinct skill, studies repeatedly find that education and training on debugging tools and techniques remain inadequate [9, 22]. For example, one study concluded that while students knew common debugging techniques, "…many students apply the techniques ineffectively or inconsistently. Testing is patchy and incomplete. Many seem unaware they should test more input than the specification outlines. Only one appeared to methodically test boundary conditions. Likewise, the use of print statements was not systematic." [22, p. 167] According to two surveys, professional programmers do indeed employ structured debugging techniques to some extent and make use of printf, assertions, and the step-wise debugger [2, 23]. At the same time, one of these studies combined the survey responses with observations of debugging practices and found that "in general, developers' theoretical knowledge

[1]You can find the materials used for the main event at zenodo.org/doi/10.5281/zenodo.7223815 [25].

and practical use of specialized debugging features are relatively shallow, just the amount that is seemingly sufficient for their debugging problems." [2] With regard to more advanced debugging tools such as back-in-time debuggers or automatic fault localization techniques, less than half of the programmers asked in one survey are aware that they exist, and even less have ever used any of these tools [23].

This partial knowledge about debugging techniques and tools may be the result of a lack of education and training on debugging [2, 22, 23]. The two surveys from above found that there is little formal education on debugging [2, 23]. In one survey, almost all participants have received debugging education at college or university, but most only received it on one occasion [23]. In both studies, participants reported that they mostly learned debugging through online resources and, if available, mentorships and similar arrangements [2, 23].

This inadequacy of education and training on debugging techniques and tools may ultimately lead to more time-consuming and frustrating debugging sessions. By promoting debugging as a distinct skill to be honed in addition to ones programming skills, debugging education may gain more attention in higher-level education and training.

2.2 Related Software Development Competitions

Using competitions to hone and promote skills has a long tradition in various areas of software development, such as algorithmic problem solving, general software engineering, game design, or cybersecurity. Two examples of prominent software development competitions are *Competitive Programming* and *Capture the Flag* contests. While many other competitions exist, we discuss *Competitive Programming* due to its prominence and *Capture the Flag* due to the similar component of understanding systems and its similar real-time nature.

Competitive Programming. A prominent competition is the format of *Competitive Programming* [4, 15]. In Competitive Programming events, contestants typically set out to solve a number of problems requiring participants to write a program. Depending on the event, the contestants are ranked according to the number of problems solved, the time it took them to solve the problems, or properties of their solution, such as the time it needs to run.

Competitive Programming events aim to give contestants an opportunity to test and improve their skills or to showcase techniques, languages, or skills. For instance, the global Competitive Programming competition *International Collegiate Programming Contest (ICPC)* advertises the contest as: "The contest fosters creativity, teamwork, and innovation in building new software programs, and enables students to test their ability to perform under pressure." [12]. The *International Conference on Functional Programming (ICFP) programming contest* states that "the goal of the contest was to allow teams from all over the world to demonstrate the superiority of their favorite programming language." [7] Even while the *ICFP event* is mostly framed as a fun event, the reports on the competition still highlight how contestants used their favorite language to create successful and elegant solutions [7, p. 406]. Both events are popular: the ICPC reported attracted 49,935 students in their 2017 edition [12] and the ICFP contests attracted between 95 to 194 teams per year between 2018 and 2021.

The competitions help to promote programming as a skill. At the same time, participants most likely do not improve their programming skills during the contests themselves but rather during the preparations for the contest. At some universities, students can take courses around Competitive Programming or featuring small programming competitions. Some competitions even have their own preparatory syllabus that participants can use to train, such as the syllabus of the *International Olympiad in Informatics (IOI)*, a computer science competition for school students [4]. Next to institutional courses, a number of books have been published about how to train for Competitive Programming contests [11, 14].

To our knowledge most of the Competitive Programming events do not feature tasks focused on debugging. Instead, the focus is on creating new programs solving given problems, sometimes using some provided libraries. Debugging is often regarded as a sub-activity in these competitions that is not encouraged explicitly. For instance, the IOI mentions debugging in the preparatory curriculum but denotes it as a necessary skill that will not be prompted explicitly by the tasks.

The SCORE contest (Student Contest on Software Engineering) [18] is a variation on Competitive Programming competitions that tries to cover software development more holistically. In the corresponding contests, teams of students work on a selected project and are judged on a variety of aspects such as the degree to which the delivered system meets the requirements, the quality of the implementation, or the way the team collaborates internally or with external stakeholders. As such, SCORE contests include debugging as an inherent activity of software development but do not judge participants according to it.

Capture the Flag. Another area of software development that features competitions is *cybersecurity*. In cybersecurity competitions, contestants generally have to install or overcome security measures in a variety of contest formats. One popular format is *Capture the Flag (CTF)* in which participants have to plant a token in or retrieve a token from an unknown network or system. In the *attack/defense* variation of CTF, teams are set against each other, with each team trying to secure a vulnerable system before attacking the other team's system. Similar to Competitive Programming, cybersecurity contests can attract hundreds of participants. For instance, in 2021, 433 teams participated in the *DEFCON CTF* qualification round. To disseminate knowledge about techniques and tools among participants, some contests, such as DEFCON CTF or Google Hackceler8, highlight and publish notable approaches or recordings of whole sessions during a contest. Next to being popular events in themselves, cybersecurity contests are also used in cybersecurity education, where students enjoy them as well as train relevant skills [16, 29].

While in Competitive Programming, contestants create new programs from scratch, contestants of a CTF contest work to understand and manipulate an existing system. Both, cybersecurity activities and debugging share this focus on existing systems. At the same time, CTF contests are not designed to explicitly prompt debugging. The inherent nature of debugging tasks and cybersecurity tasks differs in that CTF contestants are looking for previously unknown divergent behavior, while in debugging the divergent behavior is the starting point.

3 Competitive Debugging and Debugging Contests

We propose *Competitive Debugging* as a way to promote debugging as a skill and to spread knowledge about debugging techniques and tools. In *Competitive Debugging*, participants find and repair prepared failures of a software system and are rated on various scales such as the time they needed to repair the failure, the correctness of the patch, or the degree to which the underlying fault was repaired. This basic concept can be implemented in *debugging contests* that may differ in their format depending on their specific goals.

Our proposed format of a debugging contest is the result of two pilot runs (see Sections 4.1 and 4.2). We ran the resulting format with a group of ten participants, most of them graduate-level students (see Section 5). In our contest format, all participants work on the same task individually and are rated according to the correctness of their solution and their debugging speed. While, in the following sections, we propose one specific contest format, there are numerous considerations and variations points that may lead to very different formats (see Section 6).

3.1 Goals

We designed our contest format based on a set of explicit goals. The primary goal is to advertise debugging as a skill that one can learn and improve. Thus, we designed the format so that the contest attracts participants and engages them throughout the contest. For example, we did not strive to reconstruct realistic debugging scenarios, as they may occur in everyday software development but to create tasks that are feasible within the time-frame and enjoyable. Further, to motivate participants to take the time to learn and practice debugging techniques, the event should serve as a goal to work towards. Finally, we want to promote exchange among participants about debugging tools and techniques by providing a public stage for showcasing skills and a gathering to discuss them.

3.2 Contest Format

The core of our contest format is defined by three aspects: a previously unknown system, individual work on the same task at the same time, and no feedback on the rating of a patch before the submission.

We chose to let participants debug an unknown system to focus the contest on fault localization and patch generation. At the beginning of the contest, the organizers demonstrate how the software system to be debugged is used and give an overview of all modules and key classes. Participants cannot investigate the system in advance of the contest. If participants were allowed access to the system in advance, the chances of winning would depend on the time invested in program comprehension in advance of the competition. Also, by reducing the pressure to work on the system before the contest, we keep the time investment for this individual contest low which lowers the bar for people to participate.

In our contest formats, all participants individually work on the same task at the same time. Each task consists of one failure for which the participants should create and submit a patch. At the beginning of each task, the organizers present the failure. Afterwards, the participants change to their own systems and start working on the task. By working on the same task at the same time, participants have a shared experience and can compare their strategies and results among each other after the round finished. Also, by having the same tasks for all participants the results are directly comparable.

A central mechanism of our tasks is that participants have to decide on their own when the failure is repaired and when they want to submit their patch. Submitting a wrong patch results in zero points for the task (for details see Section 3.3). We do not provide ready-made automated tests, as we want participants to investigate the failure up to the point that they are confident that they understood the infection chain. As a result of this pressure to understand the failure, we hope to encourage a conscious choice of debugging techniques.

3.3 Scores and Rules

Contestants are rated according to the correctness and speed of repairing the failure. A patch is rated as either correct or wrong. If the patch is wrong, the contestant gets zero points for the task. When the patch is correct, the contestant can get up to six points, depending on how fast they solved the task. For every five minutes passed, participants receive on point less, so they start off with six points and lose one every five minutes, given that their submitted patch is correct. With this rating system, we aimed to emphasize correctness, as a wrong solution results in loosing all points for a task, while still rewarding efficient debugging techniques. Using an absolute measure allows for a clear interpretation of scores,

1. Start the game.
2. Navigate to islands until a combat is announced.
3. Choose "Stay in Formation" to start a ground combat.
4. Move one of your unit to the left of an enemy unit.

After moving your unit to the left of an enemy unit, the attack button still remains disabled. Normally, the attack button should be enabled and when you click on the button the attack should be executed on the enemy unit.

Figure 2. Description of a Competitive Debugging task.

but also requires organizers to balance tasks well, as very difficult or very easy tasks may result in very similar scores across the contestants.

To prevent shortcut solutions, we set up a few basic rules for contestants. First, contestants are not allowed to apply meta approaches, such as looking for the mechanisms installing the faults in their system or searching for the files representing the tasks. This is similar to rules in CTF contests that exclude hacking the contest server. Further, we exclude the use of any history information in this contest such as the local history or version control information, as the loading of faults is part of the local history and we did not construct a proper history of the system. Searching the Internet is not forbidden but with our task design we try to ensure that all information required to solve the task is available locally.

3.4 Scenario and Tasks

We use manually seeded faults as tasks, as we need control over the complexity of the tasks and are not interested in a realistic setting. Each task includes the steps to reproduce the failure, a description of the wrong behavior to be observed, and a description of the expected behavior (see Figure 2 for an example from our main event). As our goal is to engage participants, the tasks should prompt interesting techniques, but at the same time be solvable in 30 minutes for most participants. Further, most of the task complexity should result from the fault localization and not from understanding the task description [26]. To reduce the complexity of understanding the task description, the tasks contain complete but short failure descriptions. With regard to the faults, we limit ourselves to ordinary control flow faults, such as a flipped boolean, to keep the entry barrier low. Future contests may also feature other kinds of faults, for instance faults in concurrent parts of a system (see Section 6). Further, all tasks only contain a single fault at a single source code location, manifested as a wrong, missing, or superfluous statement.

3.5 Observing other Contestants

To encourage learning from others and make the event more fun, the contest features streams of the programming environments of some contestants. As with code katas [5], participants gain another chance to reflect on their practice by deliberately watching the practices of others. Streaming is voluntary as not to deter participants who would like to participate but are not confident enough yet to show their process. A commentator accompanies the stream and continuously points out interesting techniques and reflects on potential hypotheses and approaches of contestants.

The stream can be watched by all contestants who finished their tasks. As this would result in faster contestants to have more opportunities to watch streams than slower ones, we also decided to provide everyone a chance to observe by randomly selecting participants to skip a task and watch the stream instead. To ensure that skipping a task does not result in a disadvantage depending on the difficulty of the task, participants who skipped a task get the mean score for that task.

3.6 Curriculum

As with Competitive Programming or CTF contests, a Competitive Debugging contest is an exciting event that can motivate participants to improve their skills. To leverage this motivation, we offer a preparatory recap session and a training environment and scenario.

The preparatory session is one week in advance of the contest and covers the following topics:

- distinction between fault, failure, and infection chain
- debugging strategies: TRAFFIC [33], heuristic and systematic debugging and when to switch from the first to the second
- debugging techniques: the scientific method of debugging with explicit hypotheses and deliberate experiments, backwards and forward reasoning, input manipulation [3]
- standard debugging tools and advanced features of these tools

Further, these topics are illustrated during a collaborative debugging session of an example task. After the session, participants get a training environment that includes the example task and further training tasks.

4 Pilot Events – Setups and Insights

As there are no prior debugging contests, we first conducted two pilot events to determine the experience of participants and potential blind-spots in our initial ideas for the concept. The setups of these two pilot events differed from the setup of our main event (see Section 3 and see Section 5). In the following, we describe these initial, preliminary setups and the insights we gained from these first two debugging contests.

4.1 Contest Pilot

Our first pilot was aimed to get general feedback on the idea of a Competitive Debugging event.

Setup. The setup of this contest was similar to the setup of the main event (see Section 3), with some notable differences. For the pilot, we recruited contestants ranging from graduate level students, to PhD-students, post docs, and one professional programmer. All contestants worked in the same development environment that we prepared ahead of time. In addition, two people were purely observing the event. Once a task was completed, contestants could also join the audience until everyone was done or the time elapsed. The difficulty of the tasks had a wider spread than our final competition format (see Section 3), ranging from simple mixups of boolean values to patches that required rewriting whole methods. At the time, our infrastructure did not permit easily giving indication of performance, as such participants did not know their scores until after the event.

Insights. We conducted semi-structured interviews with contestants and observers to determine how participants experienced the event. Despite technical difficulties (interrupted streaming, crashing programming environments during one task), all participants reported that they enjoyed the event. Contestants reported that they liked the short time frame for the tasks, as it reduced the time investment and as such their fear of wasting their time. Concerning task difficulty, contestants described that their expectation on the level of difficulty was unclear. Since the first task involved a simple flipped Boolean literal, participants had a tendency to search for problems on a similar level of difficulty, even when a later task required changing a data structure. As a result, in the final competition format, we only used tasks of similar size, to avoid misleading participants. Observing other participants and receiving commentary from the organizers was generally perceived as fun, and as such more entertaining than educational. Two participants in particular began comparing observed techniques to their own and commenting while they were part of the audience.

4.2 Tool Show Contest Pilot

Our second pilot was aimed to test two variation points: a larger audience and a non-standardized development environment with custom tooling. The goal was to allow the audience to learn about the strengths and weaknesses of new programming tools by observing how contestants proficient with the respective tools would use them to solve the same task.

Setup. We recruited three contestants, each proficient with a different development tool (incremental back-in-time debugger, keyboard-controlled block-based programming environment, tool environment based on the ideas of artifacts and projections). Right before the contest we held the training session for contestants of the main event (see Section 3.6). The example task we debugged in the training session was the first task of the tool show contest, so that participants of the training session had some knowledge of what the defect was and how it could be debugged. Solutions to the other tasks were explained by the moderator to the audience just after each round began. Eight of the participants of the training session joined as the audience for the tool show contest. We also included a small variation in the tasks: this time, they involved incomplete descriptions (only one symptom was described) and failures resulting from multiple defects.

Insights. We conducted semi-structured interviews with observers in the audience and noted some initial reactions of the three contestants concerning task difficulty. All observers in the audience stated that they enjoyed the session, mentioning that the format did not have any boring or lengthy sections, even when some participants took longer to solve a task. However, observers agreed that learning about the showcased tools was difficult: understanding both a previously unknown task and tool posed a challenge, worsened by the fact that we regularly switched between contestant's streams during each task. Observers noted that commentary from moderator, both on the task and the tools were helpful to follow along. In particular, each time a unique feature of the showcased tool was used the moderator pointed it out and participants tended to cheer. These moments when unique features were used were somewhat rare, however, as we did not pay special attention to design tasks in such a way that specialized strengths of the tools became apparent. For this type of format, we also noticed the importance of ensuring a comparatively higher level of complexity in the tasks, as some participants finished some tasks within two minutes, leaving little room to see the tools in action. Nearly all audience members pointed out that they very much enjoyed knowing the first task ahead of time from the training session and were thus able to easily follow along. From this pilot, we concluded that there is interest in this observer-heavy format, but to achieve the goal of showcasing advanced tools, the contest format needs to be adapted more.

5 Main Contest – Setup and Experiences

With the main event we aimed to test our contest format for a larger group and explore how a Competitive Debugging event may influence participants' perception of and knowledge about debugging. For the main event, we recruited nine graduate students and one professional programmer who recently graduated. Of these, nine participated in the debugging training session one week prior to the contest.

Figure 3. A photography of parts of the audience watching a contestant's stream during the main contest. The dashboard is visible at the top left of the screen.

5.1 Event Setup

The event corresponds to the contest format described earlier (see Section 3) with the following detailed setup.

Programming Environment. All participants work in the same programming environment (also see Section 6), an unaltered version of the Squeak/Smalltalk environment [10, 13]. We chose this environment, as it provides a variety of debugging tools and all recruited participants are familiar with it. Squeak/Smalltalk is a live programming environment, so the whole system can be modified while it is running [27]. Participants can use the *workspace* to write scripts and execute them, *the explorer and inspector* to inspect the state of any object in the system, the *halo meta-menu*[2] to reflect about the user interface and navigate from graphical elements to the source code, and a *step-wise debugger* that also allows for arbitrary dynamic code execution and restarting the execution of method invocations. Participants are allowed to customize the environment, but not to load additional programming tools not provided by the environment. For instance they are allowed to load a window manager or their own set of shortcuts. We restrict modifications of the environment to focus on the basic techniques of debugging. As all participants use the same tools, they can more easily compare their techniques. Other contest formats may want to also allow for more advanced tools to promote such tools (see Sections 4.2 and 6).

System and Tasks. The system participants work on is a computer game. We selected a game, as the expected group of participants was likely to be familiar with the architecture of small games. Whether developers have prior experience with an application domain can have a major impact on their debugging efficiency and used techniques [6, 31]. As all expected participants have previously taken project courses

[2]A feature of the Squeak/Smalltalk environment that allows picking a user interface element via the mouse and inspecting and modifying the object representing it.

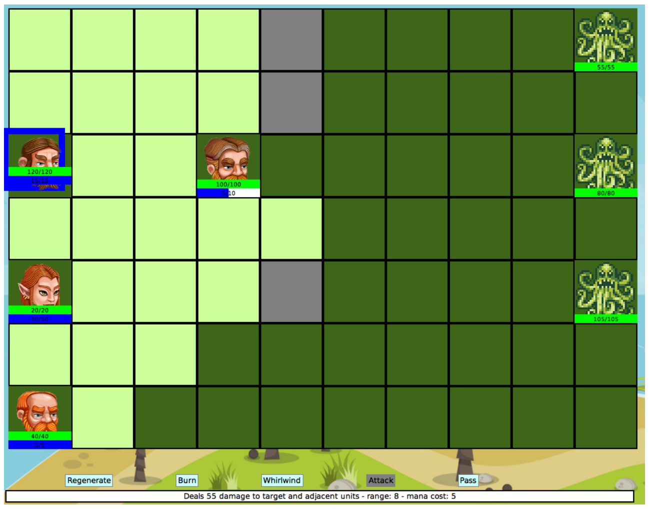

Figure 4. Screenshot of the game "Realms-of-Zaltia". Shown is the positional combat scene.

Table 1. Metrics characterizing the game "Realms-of-Zaltia".

Metric	
#Packages	5
#Classes	50
#Methods	893
#Methods / Class	17.9
LOC	4573
LOC / Method	5.1

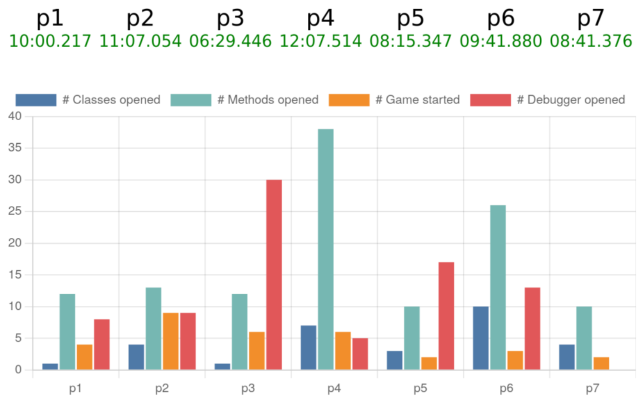

Figure 5. Screenshot of the dashboard as shown during the contest. Displayed are the statistics for each active contestant during a single task. In the top row are the total time for the task for each participant, as the round had finished. The participant names have been redacted.

with small games as projects, we expected them to have similar prior experiences with games.

We also selected a game, as games offer a variety of concerns such as event handling, state propagation, file I/O, rendering, and algorithms. This variety allows us to define a variety of different tasks.

The particular system we have selected is the game "Realms-of-Zaltia", developed by a group of undergraduate students in a course on software architecture. It is a single-player role-playing game including the following features (see Figure 4):

- basic and positional, turn-based combat
- randomly generated world maps,
- sprite-based graphics and animation,
- equipping characters with items,
- buying and selling items,
- experience and skill points,
- sound, and
- custom widgets and menu classes.

With 4573 LOC the game has an average size for a game with the aforementioned features developed during the course (for more metrics see Table 1).

Schedule and Streaming Setup. We ran one warm-up task and four tasks for the contest. During each of the four task, we had either three or two participants skip their turn and instead observe the other participants working. We had asked four participants to stream their setup and ensured that during each task we had at least two people streaming.

In addition, we instrumented the development environment to, for each contestant, report the

- number of unique touched classes,
- number of unique touched methods,
- number of times the debugger opened,
- number of times the game was restarted, and
- start and end times.

These stats were displayed in a live-updating dashboard that we displayed next to the streams (see Figure 5).

After participants completed all tasks, the organization team checked whether the submitted patches were correct, calculated the scores based on the correctness and the time taken, and awarded the average scores of each run to participants who were observers. We decided to manually judge the correctness of the patches, as we were not confident yet, that we understood the potential variety of submitted patches. No participant had submitted incorrect solutions during this run. The leaderboard including final scores was then sent out via mail shortly after the event.

5.2 Participants' Experience

To determine the general merit of debugging contests and avenues for improvement, we asked participants about their experience during the contest.

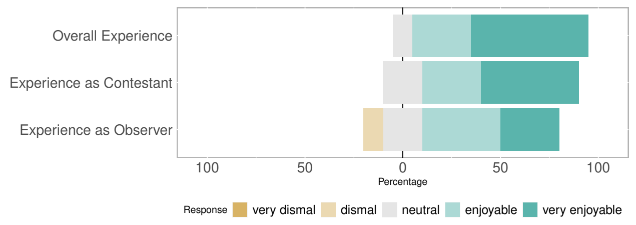

Figure 6. Results of the questionnaire questions on participant's experience. Participants rated their experience as observers less enjoyable than their experience as contestants.

After the contest was over, we asked all participants to fill out a short questionnaire containing ten likert-scale statements about their experience and how the event influenced their perception of debugging (see Section 5.2 below). We then proceeded to semi-structured interviews with the participants. The questions should help start the conversation with participants about their learnings from being contestants or observers, how the contest influenced their perception of debugging, how the training influenced their approach to the event, which formats they would be interested in in the future, how they experienced the competitiveness, and finally their general experience.

We summarized the insights from the interviews according to major, recurring themes.

Questionnaire Results. Overall participants very much enjoyed the contest, a result also reflected in the interviews (see Figure 6). With regard to whether how they enjoyed being a contestant or an observer, participants reported to have enjoyed being an observer slightly less than being a contestant.

According to the questionnaire responses, participants did not have a strong impression of having learned something about debugging at the conference (see Figure 7). However, the contest seemingly achieved its purpose of sparking interest about debugging and prompting participants to regard debugging as a skill to be improved. Almost all participants reported to have reflected about their debugging process during the contest and most participants reported to be motivated to learn more about debugging techniques and tools.

Experience as a Contestant. All contestants reported that they generally enjoyed the whole event and all stated that they would be interested in taking part in other debugging contests.

The fact that the event was a contest and that they had to work under time-pressure led to various impressions. For some participants the competitive pressure to debug quickly was exciting and motivating. One contestant put it as: "I liked that there is something that makes one nervous but

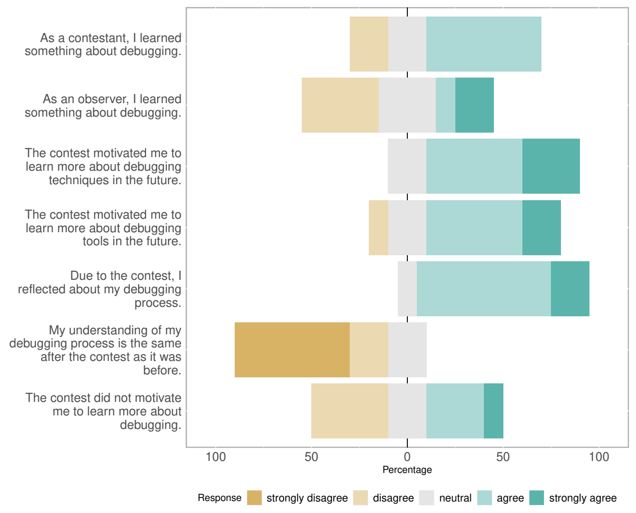

Figure 7. Results of the questionnaire questions on participant perception of how the contest influenced how they see debugging. The bottom-most questions are inverted questions. Participants did not have a strong impression that they learned something about debugging during the event. At the same time, due to the contest they reported to have reflected about their process and felt motivated to learn more about debugging.

is actually without consequences."[3] At the same time, several participants who only saw a small chance of winning reported that they mostly tried to keep face, but even among this group all participants enjoyed the event.

Difficulty of the tasks. During the interviews, the difficulty of the tasks came up regularly. Contestants liked that all tasks were generally solvable within the time-frame. One participant reported that they got the impression that there was not much to learn about debugging anymore, which might be the result of the small scope and basic characteristics of the defects. Finally, for some participants the expectations on the task selection led to some meta-gaming. One participant reported that "because I knew in advance that the bugs can not be too complicated, I reached the solutions more quickly . . . this felt like cheating sometimes".

Changed Perception of Debugging. Several but not all participants reported that they inspected their own debugging activity more consciously during the contest. Insights ranged from minor workflow improvements to general shifts in perception. One participant reported that the contest helped them realize how to leverage live programming capabilities to reduce the time to reproduce a failure by keeping the game running. Another contestant reported that they discovered a number of unnecessary or unproductive activities in their debugging process. Finally, one participant even reported a general shift in their perception of their debugging process, as they realized that they very often try to find defects solely by reading code and seldom use the debugger, a strategy they want to re-consider in the future. They explained that they realized this after seeing the statistics in the dashboard.

[3]All quotes are translated from the participants' native tongue (German) to English.

Relation to Training Session. Several other participants related their learnings during the contest to the training session. Many mentioned that they started reflecting about their hypotheses during their debugging progress or at least recognized their hypotheses as such. One participant mentioned that they attended the training for the contest, but managed to already apply the techniques successfully in their everyday work in the week up to the contest. Others mentioned that they planned to use input manipulation as a techniques, but did not manage to apply it successfully during the contest. Several participants with experience with Competitive Programming reported that they would have liked even more training, as they felt that this is where most of the learning occurs in Competitive Programming.

Experience as an Observer. The opinions on observing other contestants was more mixed than in the pilot runs. One participant found it outright boring. Others had trouble following along, even though we kept the stream with one participant from start to finish, potentially because they only started watching half-way through the task. One problem of this event was that due to the voluntary streaming there were only four streams and as soon as these contestants completed the tasks, there were no more streams to watch. Several participants would have liked to keep on observing other contestants. The dashboard bridged the gap to some extent and allowed for some more commentary but was ultimately less interesting than a full stream. While only one participant reported learning a new technique from watching other contestants (conditional breakpoints), many participants still found it interesting to see others using the tools. They also liked the stream to judge their own skill level in relation to others. Finally, several participants reported that they used the phases for watching the streams not to observe others, but to ask the other contestants how they approached the task. They found these discussions so useful that they also suggested to allocate more time for discussing solutions.

Future Formats. All participants were interested in future editions of the event. Some even suggested larger formats on their own that would cover a more extensive curriculum and include more difficult defects. Other enjoyed the fast pace of the event and would like to participate in another event with similar tasks.

6 Considerations

Our proposed contest format is geared towards engaging participants and promoting debugging as a skill. While designing the Competitive Debugging format, we encountered a number of considerations, which we describe in the following.

Depending on how organizers decide on these considerations, future contests may take on various forms. For example, a contest may aim to showcase long-term strategies and advanced tool usage, and may therefore feature teams working on difficult defects sourced from a large open-source project. Another contest may aim to promote techniques for specific kinds of defects such as defects related to concurrency or memory corruption, and may therefore only feature defects of that particular category. Finally, an event may focus on the audience instead of the contestants and therefore feature a small number of defects that are interactively debugged with the audience in advance of watching few expert debuggers trying to repair them live.

Programming Language. In the Competitive Programming and CTF formats, programming languages can typically be chosen freely, as the tasks are mostly concerned with program output. In Competitive Debugging, contrarily, the task concerns the system code itself. As it is non-trivial to create a comparable debugging scenario in two different programming languages, the choice of language will be a major point deciding which communities will be addressed by the event.

Tools. Unlike with programming languages, a contest may allow contestants to use different tools and programming environments and still pose a comparable challenge. Of course, if all contestants are using the same tools, it becomes easier to compare their approaches. However, with an open choice of tools, there may be greater learning opportunity for observers, as they can learn about the different strengths of various tools.

Choice of System. When choosing a system to be debugged, the application domain and the size and inner quality of the system can influence the character of the competition.

As described earlier, the application domain has a major influence on the debugging strategies and techniques. In our contest, all participants were familiar with small games as an application domain. In more heterogeneous groups of participants, participants will have more diverse experiences with application domains. In this case, organizers may announce the application domain in advance to enable participants to decide whether they want to participate or enable them to catch up about basic principles of the application domain.

An interesting consideration arose due to the fact that we used a system written by undergraduate students: participants tended to assume they were given generally well-written code, which was not always true. As such, some tasks increased significantly in complexity as bad naming choices, surprising implementations, or mixing of layers of abstraction led participants to discard or pose incorrect hypotheses during debugging.

Tasks. Many participants of our events mentioned that simultaneously working on a single task resulted in interesting and enjoyable aspects such as post-task discussions and the possibility to directly compare approaches. At the same time this schedule of releasing one task at a time constrains the size of tasks, as more complex tasks may result in a larger spread between contestants' results and thus more waiting time. Alternatively, participants may receive all tasks at once, similar to how Competitive Programming contests are organized. This may be particularly interesting in settings with larger systems and teams competing against each others, especially when confronted with a large number of tasks that require teams to estimate effort and coordinate their approach.

Characteristics of Faults. The difficulty of the tasks mainly depends on the characteristics of the faults. In the following, we briefly describe some factors we considered when designing our tasks. For one, there can be just a single or multiple symptoms of the fault. Additionally, contestants tended to be more confused or unsure when the task description was not exhaustively listing all symptoms. Similarly, there may only be a single or multiple places that require patching. In particular, if there are multiple places, does the task description hint at all symptoms that occur or would the participants have to anticipate a corner case and test it themselves to provide a correct solution? The type of change may range from changing a single token, such as a literal or a property name, over deleting or adding a statement, to proper refactorings, for instance changing a data structure and optionally adapting identifier names accordingly.

Reproducing and inspecting the bug may also be challenging, for example if the bug arises due to concurrency or does not occur when in a debugger. If working in a live system such as Squeak/Smalltalk, bugs that crash the development environment often require specific strategies to investigate as well. Lastly, bugs may be designed to entice use of specific tooling, such as plots when a value in a control system is not developing over time as desired.

Individuals vs. Teams. Debugging is commonly done alone or in pairs. If allowed to work in pairs, participants may experience the contest as more fun, as there are more social interactions. Additionally, it may help participants explicating their process, as they have to explain their thoughts to their partner. An often mentioned aspect was also the wish of observers to be able to follow along with what the contestants are actually thinking. Working in teams may thus enable getting insights into some thought processes if audio was transmitted alongside the video.

Integrating Program Comprehension. Previous knowledge about the system to be debugged determines what kind of hypotheses participants can generate. In our format, we balanced previous knowledge about the system by confronting participants with a completely new system and only providing a small introduction just before the contest. To emphasize program comprehension techniques and tools, the contest may have an initial phase during which participants may learn as much about an unaltered version of the system as possible before the tasks begin where bugs are injected in the system.

Training. Even our small training and recapitulation one week before the main event had a considerable influence on participants and their reported behavior during debugging. Participants stated that they were looking for opportunities to apply learned techniques and consciously began formulating hypotheses rather than following their usual, often unstructured, approach. In Competitive Programming, the training ahead of a contest is an integral part of the competitions. Similarly, debugging contests may feature regular training and teaching sessions in advance of the event. In particular, if focus is put on a specific domain or tool participants could benefit from a preparation that introduces common failures or approaches.

Scaling the Event. Organizers might have to reconsider some of our decisions when scaling a debugging contest to a setting with more than twenty participants.

First of all, task management, including scoring, would need to be fully automated to allow for timely feedback on task results. We used manual scoring for our main event, as we were yet unsure of the variety of potential approaches to the task and preferred to manually decide on the correctness of a task. This is infeasible for larger audiences and may be replaced with a set of unit tests to be executed on submission of the patch. Further, our current scoring system might be unsuitable for larger audiences, as it might not result in a sufficient spread in the scores to get distinct positions per participant. This might be acceptable in university teaching situations where a general measure of skill might suffice, but may be less motivating in other settings.

Second, in a contest, that includes streaming of participants' screens, a higher number of observers may result in higher pressure for the participants who stream, which may in turn deter participants from streaming.

Finally, a large number of participants results in a mundane logistic challenge. For our events, we used two rooms: in one room participants' worked on the task in silence, and in the other participants' watched the streams and chatted. Participants left the working room quietly to not disturb the remaining contestants. This setup might not be possible with larger numbers of participants.

Observing. With our format, we also aimed to create an informative and enjoyable format for observers. Based on the insights from our three contests, we are not yet sure what an ideal format may look like.

A first consideration was the frequency of switching between streams. Staying focused on a single contestant per task allowed observers to more easily understand and follow along with their thought process. However, there were in parts situations where contestants got stuck and spent time not interacting with the development environment at all. Switching to another contestant may allow to fill these gaps more easily. Additionally, when switching streams, observers can get a broader impression of the various approaches to the same task. Instead, with a single focus, contestants who completed their tasks and had not been observed typically gave a quick summary of their approach in the post-task discussion.

We found that, especially in a remote or hybrid setting, creating smaller groups with a moderator each, benefited lively discussions among observers. The moderator had the important function of providing observers with a sense of the actual effectiveness of contestants' steps through commentary, as the moderators knew the location of the defect and had at least moderate familiarity with the structure of the system. If given more time to prepare, observers during our second pilot also reported that they particularly enjoyed observing contestants solve the task they had previously solved themselves. This may be integrated in the format either by prefixing each task with a detailed walk-through by the moderator or by giving observers access to the tasks they will be observing before the event.

The dashboard (see Figure 5) allowed observers to get an overview of how the contestants were generally performing. The collected statistics were useful for some meta commentary (for instance, "p1 has not opened the debugger once to solve this task") but were only of limited use to see usage patterns, as they were limited to "open" events. A better picture of participants' activities may be gained by sending events only after spending some time with an opened artifact or reporting active phases in each tool or artifact. Additionally, to gain a fast overview of the progress of participants, it may be worthwhile to create a metric that roughly signifies the semantic distance between the current artifacts a contestant inspects and the location of the bug. For example, this may be a number of steps in a pre-recorded call trace.

7 Conclusion

While debugging is a central activity of software development, debugging is seldom taught and trained extensively. By introducing *Competitive Debugging* we set out to highlight debugging as a skill and to provide a platform for software developers to discuss and exchange debugging techniques and tools to improve their skills. The basic mechanism of who finds the correct patch for a defect in the shortest amount of time can be implemented in various different contest formats. We propose a format designed to attract and engage participants to motivate them to learn more about debugging. In summary, based on two pilot events and a main event, we found that participants were highly engaged during the format, more so as contestants than as observers, and stated that they reflected on their approach to debugging due to the contest. In addition, we list a number considerations for future formats to help organizers of future debugging contests find suitable tasks or adapt the observers' experience to their setting.

Debugging contests may have the potential to help developers refine their debugging techniques and spread the word about advanced debugging tools and thereby, in the end, help software developers to be more engaged and productive when debugging.

Acknowledgments

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 449591262. We also gratefully acknowledge the financial support of HPI's Research School[4] and the Hasso Plattner Design Thinking Research Program[5].

References

[1] Marzieh Ahmadzadeh, Dave Elliman, and Colin Higgins. 2005. An analysis of patterns of debugging among novice computer science students. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2005, Caparica, Portugal, June 27-29, 2005*, José C. Cunha, William M. Fleischman, Viera K. Proulx, and João Lourenço (Eds.). ACM, 84–88. https://doi.org/10.1145/1067445.1067472

[2] Moritz Beller, Niels Spruit, Diomidis Spinellis, and Andy Zaidman. 2018. On the dichotomy of debugging behavior among programmers. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (Eds.). ACM, 572–583. https://doi.org/10.1145/3180155.3180175

[3] Marcel Böhme, Ezekiel O. Soremekun, Sudipta Chattopadhyay, Emamurho Ugherughe, and Andreas Zeller. 2017. Where is the bug and how is it fixed? an experiment with practitioners. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, Eric Bodden, Wilhelm Schäfer, Arie van Deursen, and Andrea Zisman (Eds.). ACM, 117–128. https://doi.org/10.1145/3106237.3106255

[4] Benjamin Burton. 2022. Foreword to Olympiads in Informatics Vol. 16. *Olympiads in Informatics* 16 (2022), 1–2. https://doi.org/10.15388/ioi.2022.00

[5] Thomas Clavier, Alexis Benoist, Emmanuel Gaillot, Olivier Albiez, and Étienne Charignon. 2022. *WhatIsCodingDojo*. http://web.archive.org/web/20221018153900/https://codingdojo.org/practices/WhatIsCodingDojo/

[6] Françoise Détienne. 2001. *Software design cognitive aspects.* Springer. http://www.springer.com/computer/swe/book/978-1-85233-253-2

[7] Eelco Dolstra, Jurriaan Hage, Bastiaan Heeren, Stefan Holdermans, Johan Jeuring, Andres Löh, Clara Löh, Arie Middelkoop, Alexey Rodriguez, and John van Schie. 2008. Report on the tenth ICFP programming contest. In *Proceeding of the 13th ACM SIGPLAN international conference on Functional programming, ICFP 2008, Victoria, BC, Canada, September 20-28, 2008*, James Hook and Peter Thiemann (Eds.). ACM, 397–408. https://doi.org/10.1145/1411204.1411259

[8] Marc Eisenstadt. 1997. My hairiest bug war stories. *Commun. ACM* 40, 4 (1997), 30–37.

[9] Sue Fitzgerald, Gary Lewandowski, Renée McCauley, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers. *Comput. Sci. Educ.* 18, 2 (2008), 93–116. https://doi.org/10.1080/08993400802114508

[10] Adele Goldberg and David Robson. 1983. *Smalltalk-80: The Language and Its Implementation.* Addison-Wesley.

[11] Steven Halim and Felix Halim. 2013. *Competitive Programming, Third Edition.* Lulu.

[12] icpc.foundation. 2022. *ICPC Contest 2021 Page.* http://web.archive.org/web/20220704072158/https://icpc.global/regionals/abouticpc

[13] Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan C. Kay. 1997. Back to the Future: The Story of Squeak - A Usable Smalltalk Written in Itself. In *Proceedings of the 1997 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages & Applications, OOPSLA 1997, Atlanta, Georgia, October 5-9, 1997*, Mary E. S. Loomis, Toby Bloom, and A. Michael Berman (Eds.). ACM, 318–326. https://doi.org/10.1145/263698.263754

[14] Antti Laaksonen. 2020. *Guide to Competitive Programming.* Springer International Publishing. https://doi.org/10.1007/978-3-030-39357-1

[15] Alex Lang and Jasper Van der Jeugt. 2021. *ICFP Contest 2021.* http://web.archive.org/web/20220507121209/https://icfpcontest2021.github.io/

[16] Kees Leune and Salvatore J. Petrilli Jr. 2017. Using Capture-the-Flag to Enhance the Effectiveness of Cybersecurity Education. In *Proceedings of the 18th Annual Conference on Information Technology Education and the 6th Annual Conference on Research in Information Technology, SIGITE/RIIT 2017, Rochester, New York, USA, October 4-7, 2017*, Stephen J. Zilora, Tom Ayers, and Daniel S. Bogaard (Eds.). ACM, 47–52. https://doi.org/10.1145/3125659.3125686

[17] Bil Lewis. 2003. Debugging Backwards in Time, In Proceedings of the Fifth International Workshop on Automated Debugging (AADE-BUG 2003). *CoRR* cs.SE/0310016. http://arxiv.org/abs/cs/0310016

[18] Dino Mandrioli, Stephen Fickas, Carlo A. Furia, Mehdi Jazayeri, Matteo Rossi, and Michal Young. 2010. SCORE: the first student contest on software engineering. *ACM SIGSOFT Softw. Eng. Notes* 35, 4 (2010), 24–30. https://doi.org/10.1145/1811226.1811240

[19] Renée McCauley, Sue Fitzgerald, Gary Lewandowski, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: a review of the literature from an educational perspective. *Comput. Sci. Educ.* 18, 2 (2008), 67–92. https://doi.org/10.1080/08993400802114581

[20] Tilman Michaeli and Ralf Romeike. 2019. Current Status and Perspectives of Debugging in the K12 Classroom: A Qualitative Study. In *IEEE Global Engineering Education Conference, EDUCON 2019, Dubai, United Arab Emirates, April 8-11, 2019*, Alaa K. Ashmawy and Sebastian Schreiter (Eds.). IEEE, 1030–1038. https://doi.org/10.1109/EDUCON.2019.8725282

[21] Tilman Michaeli and Ralf Romeike. 2019. Improving Debugging Skills in the Classroom: The Effects of Teaching a Systematic Debugging Process. In *Proceedings of the 14th Workshop in Primary and Secondary Computing Education, WiPSCE 2019, Glasgow, Scotland, UK, October 23-25, 2019.* ACM, 15:1–15:7. https://doi.org/10.1145/3361721.3361724

[22] Laurie Murphy, Gary Lewandowski, Renée McCauley, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: the good, the bad, and the quirky – a qualitative analysis of novices' strategies. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer*

[4] https://hpi.de/en/research/research-school.html

[5] https://hpi.de/en/dtrp/

Science Education, SIGCSE 2008, Portland, OR, USA, March 12-15, 2008, J. D. Dougherty, Susan H. Rodger, Sue Fitzgerald, and Mark Guzdial (Eds.). ACM, 163–167. https://doi.org/10.1145/1352135.1352191

[23] Michael Perscheid, Benjamin Siegmund, Marcel Taeumel, and Robert Hirschfeld. 2017. Studying the advancement in debugging practice of professional software developers. *Softw. Qual. J.* 25, 1 (2017), 83–110. https://doi.org/10.1007/s11219-015-9294-2

[24] Guillaume Pothier and Éric Tanter. 2009. Back to the Future: Omniscient Debugging. *IEEE Software* 26, 6 (2009), 78–85. https://doi.org/10.1109/MS.2009.169

[25] Patrick Rein, Tom Beckmann, Leonard Geier, Toni Mattis, and Robert Hirschfeld. 2022. Materials for Conducting Debugging Contests. https://doi.org/10.5281/zenodo.7223815

[26] Patrick Rein, Tom Beckmann, Toni Mattis, and Robert Hirschfeld. 2022. Toward Understanding Task Complexity in Maintenance-based Studies of Programming Tools. In *Proceedings of the Programming Experience 2022 (PX/22) Workshop, Porto, Portugal, March 21, 2022.* ACM, 1–8. https://doi.org/10.1145/2984380.2984381

[27] Patrick Rein, Stefan Ramson, Jens Lincke, Robert Hirschfeld, and Tobias Pape. 2019. Exploratory and Live, Programming and Coding - A Literature Study Comparing Perspectives on Liveness. *Art Sci. Eng. Program.* 3, 1 (2019), 1. https://doi.org/10.22152/programming-journal.org/2019/3/1

[28] RTI. 2002. *The Economic Impacts of Inadequate Infrastructure for Software Testing*. Technical Report. National Institute of Standards & Technology. 309 pages.

[29] Valdemar Švábenský, Pavel Celeda, Jan Vykopal, and Silvia Brisáková. 2021. Cybersecurity knowledge and skills taught in capture the flag challenges. *Comput. Secur.* 102 (2021), 102154. https://doi.org/10.1016/j.cose.2020.102154

[30] Radhika D. Venkatasubramanyam and Sowmya G. R. 2014. Why is dynamic analysis not used as extensively as static analysis: an industrial study. In *1st International Workshop on Software Engineering Research and Industrial Practices, SER&IPs 2014, Hyderabad, India, June 1, 2014*, Rakesh Shukla, Anjaneyulu Pasala, and Srinivas Padmanabhuni (Eds.). ACM, 24–33. https://doi.org/10.1145/2593850.2593855

[31] Anneliese von Mayrhauser and A. Marie Vans. 1997. Program understanding behavior during debugging of large scale software. In *Papers presented at the Seventh Workshop on Empirical Studies of Programmers, ESP 1997, Alexandria, Virginia, USA, 1997*, Susan Wiedenbeck and Jean Scholtz (Eds.). ACM, 157–179. https://doi.org/10.1145/266399.266414

[32] W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. A Survey on Software Fault Localization. *IEEE Trans. Software Eng.* 42, 8 (2016), 707–740. https://doi.org/10.1109/TSE.2016.2521368

[33] Andreas Zeller. 2009. *Why Programs Fail - A Guide to Systematic Debugging, 2nd Edition.* Academic Press. http://store.elsevier.com/product.jsp?isbn=9780123745156&pagename=search

Received 2022-07-12; accepted 2022-10-02