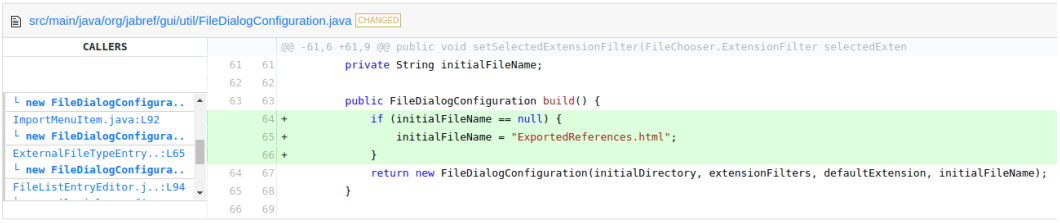

Figure 5.1: Changes to 'FileDialogConfiguration.java' in task 2, in CHANGEVIZ

The group of participants was divided in two: the *control group* using the standard GitHub interface to perform the review tasks and the *treatment group*, which was instead required to use our own code review environment. The candidates were equally divided into the two groups. The surveys presented to each group can be found in their entirety in A.5 and A.4, respectively.

In this context we define a defect as *external* if it exists because of the propagation of the side-effects of the changes outside of the changeset due to callgraph or class hierarchy relationships, as opposed to an *internal* defect which lies directly within the changeset and, therefore, can be noticed without navigating its context.

First, participants were asked a number of demographic questions about their usual programming and code review practices. Then, participants of the treatment group were allowed 5 minutes to familiarise themselves with the interface of CHANGEVIZ on an unrelated pull request. Finally, participants of both groups were asked to perform the following code review tasks:

- **Task 1**: The review of a pull request containing an *internal* defect, which should supposedly lead to similar success rates and completion times for both groups.

  For this task a time limit of 10 minutes was given.

- **Task 2**: The review of a pull request containing an *external* defect, which should highlight the capabilities of our tool and therefore lead to a higher success rate and shorter task completion times for the treatment group.

  For this task a time limit of 25 minutes was given, and the participants were given a hint after 15 minutes from the start, had they not yet found the defect in the code. The hint was the following: *think about the side-effects of the changes in 'FileDialogConfiguration.java'*.

  In fact the defect consists in the unwanted propagation of side-effects due to the changes to the method 'build()' of this file, which, as shown in Figure 5.1, has a number of callers which are affected by the change, although they are not related to the intent of the pull request.

  When participants of the treatment group did not manage to find the defect within the given time limit, we stopped the experiment and asked what would have allowed them to discover it, and collected their comments.

The pull requests are modifications to the version 4.3.1 of the JabRef[1] code base, a Java reference management software. We chose this project because it is reasonably complex (120.000 lines of code) and is actively developed on GitHub, supporting the idea of using CHANGEVIZ as an alternative to the standard GitHub interface.

For each task, a description of the pull request was given, explaining its rationale. Participants were also requested not to focus on code style and syntax issues, as the defects had a functional nature (i.e., they affected the functionality of the application).

The hardware setup for the experiment was a 15" laptop with a 1920x1080 pixel screen resolution, an Intel i7-7700HQ CPU and 16GB of RAM. The review environments were accessed using the Google Chrome browser version 69 running on the Ubuntu 17.10 operating system.

5.2 Quantitative Results

Out of all the 10 participants, 6 were postgraduate students, 1 was a PhD student, and 3 were professional software developers. All the interviewees were familiar with the GitHub pull request review interface, and none of them was familiar with JabRef. Only two of the students had professional programming experience, but most of them had at least 2 years of Java programming experience. The majority of the interviewees programmed at least once a day, and all but two of them performed code reviews at least once a month.

5.2.1 Task 1: Internal Defect

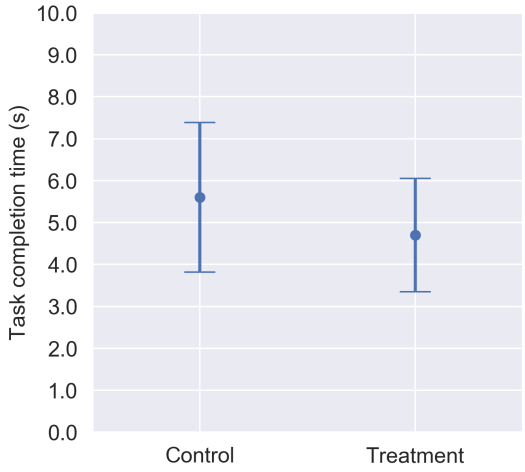

Figure 5.2: Task 1 completion times, per group

[1] http://www.jabref.org/

All the participants of both groups were able to find the defect in the first task. In terms of review success rate, therefore, we have no evidence that CHANGEVIZ performs in a worse way than the GitHub interface, for an internal defect.

The candidates of the control group had an average completion time of 5m36s (STD = 1m47s) while the ones in the treatment group had an average completion time of 4m42s (STD = 1m21s), as can be seen in Figure 5.2.

The t-test does not lead to the rejection of the null hypothesis of equal population means for the two groups ($t = 0.90$, $p = 0.39$). The normality assumption of the t-test is challenged, given the reduced sample size, but the Shapiro-Wilk normality test does not result in the rejection of the null hypothesis of a normally distributed population for each group at a $p = 0.05$ significance level, suggesting that the t-test is still appropriate. The results therefore support our conjecture that the completion time does not change significantly between the two interfaces for this type of task.

5.2.2 Task 2: External Defect

	Control group	Treatment group
Defect found	0	2
Defect not found	6	4

Table 5.1: Task 2 results, per group

Table 5.1 displays the results for the second task. None of the candidates using the GitHub interface to perform the review were able to find the defect, while only two candidates who used our tool managed to find it. Fisher's exact test for the contingency table yields $p = 0.45$, failing to detect a significant association between the groups and the task outcomes.

The two candidates who successfully completed the task, both professional developers with more than 5 years of Java programming experience, did so in 13m30s and 23m respectively, with the latter being given the hint after 15 minutes.

5.3 Qualitative Results and Discussion

The results for the first task fit with our expectations, showing equal success rates and (variations due to the reduced sample size aside) completion times for both the groups using the GitHub interface and CHANGEVIZ. This supports our hypothesis of a feature parity of the two interfaces for this type of task. Even though for the first task the usage of the sidebars provided by our tool was not necessary to find the defect (as it was in the changeset itself), most participants of the treatment group tried immediately to use them, thinking that they would have been crucial to complete this task. After a brief experimentation with the sidebars to navigate the context and get an overview of it, though, they did not seem to find any interesting

portion of code to analyse in detail, and therefore quickly went back to focusing on the main changeset as they would have normally done with the GitHub interface. As already noted, this initial context exploration did not negatively affect the average completion time, if compared to the control group. We suppose that this behaviour was caused by the fact that developers were new to ChangeViz, and being used to the GitHub pull request view, they felt the urge to make use of the navigation features offered by our interface, which they just learned about during the 5 minutes introduction to the tool.

In the second task, no participant of the control group was able to find the defect. As GitHub's review interface does not provide any support to navigate the context of the changed code, they focused only on the changeset initially, and no one tried to explore other classes in the code base. When they were given the hint after 15 minutes, some of them tried to understand how the changes to 'FileDialogConfiguration.java' and its 'build()' method affected other classes in the code base, but they quickly understood that this is a difficult task to perform on GitHub. Not having any 'Find References'-like feature available, they tried to make use of the search functionality to identify the usages of the 'FileDialogConfiguration' class, discovering that GitHub does not offer a full-text search feature for the code base of a repository at a specific commit. Realising this was not an option, they went back to the changeset and tried again to identify the defect therein, with no success.

On the other hand, two developers of the treatment group were able to find the defect. These were professional developers with significant experience in professional software development (3 to 5 and 6 to 10 years, respectively) and Java (6 to 10 years). Their extensive programming experience (compared to the other participants) may have allowed them to understand more quickly the structure of the modified classes and therefore reason about the code in more high-level terms. Most of the other developers, especially the ones with less Java programming experience, initially spent a significant amount of time trying to make sense out of the way that the code was written, being less familiar with the language and its design patterns, including the 'Builder' pattern (used within the modified files). A professional developer, with a similar amount of experience as the two participants who successfully completed the second task, was part of the control group; despite his familiarity with the language, he did not find the defect. This supports the hypothesis that the features offered by our tool are effective in performing this type of task.

We asked the developers of the treatment group who failed the second task what would have helped them to notice it. All of them, after being given the detailed description of the defect, remarked that it was reasonably easy to spot using our tool's features, being astonished not to have found it during the experiment. Surprisingly, most of them did not make a significant use of the left sidebar to analyse the side-effects of the changes, even after being told to focus on that aspect with the 15-minutes-mark hint. Based on their navigation behaviour during the experiment and their feedback, we suppose that developers got overwhelmed by the

amount of information displayed in a way they were not used to, not having had enough time to get accustomed to the tool's novel layout. We conjecture that this problem could be mitigated once developers gain experience with CHANGEVIZ and make it part of their workflow. Among the common remarks there was the request for a better visual connection between the callers and callees in the sidebars and the method definitions and method calls in the changeset, respectively. This could also encourage developers to make more use of the sidebars' features. A developer suggested to make each method call in the changeset clickable, replicating the behaviour of the click on a callee in the sidebar, which opens the class containing the called method definition and highlights it. Another developer remarked that a separation between the test and non-test classes in the callers sidebar would be ideal, displaying first the non-test ones; the rationale in this case is that unwanted side-effects in the test files are less important, as they should already be highlighted by the execution of the test suite.

5.3.1 Threats to Validity

In this section, we present the threats to the validity of our experiments.

Subjects. The reduced sample size threatens the results of our experiment. We tried to mitigate this limitation by including participants with a wide range of programming backgrounds, in an attempt to improve the representativeness of the subjects and the generalisability of our results.

Tasks. We created both the pull requests that the participants had to review in their tasks, which may have been to the advantage of our tool. We mitigated this by describing the rationale of the pull requests and making them available online, through the links in A.5 and A.4. Given the large number of participants who did not manage to solve the second task, another threat is possibly represented by a too short time given to complete it. The time limit, though, was decided based on the outcome of a pre-test, and no participant made any remark about needing more time when we asked them what would have helped them to solve the task, at the end of the experiment.

Training. Only the participants of the treatment group received an initial 5-minutes training, since we made sure that everyone was already familiar with the GitHub pull request interface. Looking at the task results, a longer training could have been helpful. However long the initial training would have been, though, it could not have gotten the participants as familiar with CHANGEVIZ as they were with the GitHub interface, given that they used the latter for a vastly larger period of time.