Dear editors, dear reviewers,

Thank you very much for the helpful suggestions.

Since the initial submission, we have continued our user study an increased the number of participants to 24.

We agree that having more bugs in the study would be desirable, but we found it difficult to find bugs that can be located in a short time with no prior knowledge of the code, so we rather focused on getting a good number of measurements for each bug. All bugs are real world bugs and representative for different classes of bugs and the core task of following infection chains does not vary too much unless when coding in specialized frameworks, which we consider out of scope for now.

Below are detailed responses to each remark of the reviewers.

Best regards,
Arian Treffer

Reviewer #1:

The overall novelty/contribution of this work seems to be skimpy given the previous version of this work by the same authors [7]. The paper claims as a major contribution introducing a new dynamic slicing algorithm, yet the algorithm (as presented in Listing 2) is quite trivial given the use of Soot for static dependence analysis.

We clarified that the novelty is in supporting iterative refinements.

Also, the dynamic slicing algorithm works by projecting static dependencies into execution trace (events) thus is not a precise algorithm. However, the authors did not present results on the precision of the algorithm, nor did they report the sizes of resulting dynamic slices. It is well known that dynamic slicing (esp. with the used imprecise algorithm) can typically produce overwhelmingly sized slices that impede their practical adoption when applied to real-world software systems.

Yes, this was an error in the description of the algorithm. We fixed this in lines 264ff and in listing 2. There is an intermediate step where we compare the static analysis results against the event database to select only the dependencies that are relevant at the respective execution of the instruction.

The evaluation is very weak in several aspects. First, the goal of the presented approach is to make dynamic slicing practically useful, yet the evaluation study only involves one experimental subject with respect to only a few test cases. Both the subject and test cases used in the study were unclear in terms of their characteristics (code size, number of test cases, etc.).

We added more details about the library and the test cases in section 5.2.1.

Second, both the subject and participants are not really close to those in practical settings: it is not clear that if the subject is widely used in practice, while the participants are mostly not real-world software development professionals.

We recruited more professionals for our study and added a more detailed discussion of the subgroups in lines 551ff and table 1.

The performance of Slice Navigator may not be really ready for practical adoption. As the authors acknowledged, waiting for a few seconds could interrupt the workflow of developers in debugging. The time for computing/updating slices with the presented tool would cost can be up to 10 seconds when less than 100K events occurred during program run time (which is pretty common in real-world software systems).

We clarified the discussion of this issue at the end of section 5.1. To summarize, the algorithm works in such a way that developers can begin working with an intermediate result almost immediately while the slicer works in the background.

Other benefits of Slice Navigator are not substantial over existing approaches. The evaluation results show that with the proposed approach developers can still feel lost as often and even take longer time to recover from being lost. The only merits seem to be reducing time on reading irrelevant code. The participants also report quite a few drawbacks of Slice Navigator, seemingly more than its benefits.

We clarified section 5.2.4 and the future work to address these concerns. To summarize: participants expected getting lost in time to be a lesser problem with more familiar code. Most reported drawbacks are reasonable requests for UI improvements. Despite those problems participants performed better with the Slice Navigator.

Important details are missing. The dynamic slicer used by Slice Navigator is mainly built on the static dependencies. It is not clear how and to what extent the static analysis overcomes well known difficulties in computing the static dependencies, such as reflection and pointer aliasing. Unsound static analysis would lead to missing instructions (where the fault locations may be) in the dynamic slices.

We added an explanation how we handle reflection in lines 259ff. We do not analyze reference aliasing because we can use runtime data to select the correct dependency from all possible locations. This is similar to the approach by Zhang et al. in “Precise dynamic slicing algorithms”.

Minor issues:
-The paper is generally readable but would need a careful pass of proofreading to clear off rough grammars (e.g., typos and broken sentences)---for instance, the first sentence of 4.2 is broken; in last paragraph of 5.2.1, "where"->"were".

Those and other style and grammar errors were fixed.

-In 3.1., 'dynamic dependence graph' is neither defined nor referenced through other works.
-In 3.3., the three dependence types are not formally defined, although the authors state that they are 'formally' defined. The current definitions are too vague and anecdotal.

We added more formal definitions and references to related work in both cases.

Reviewer #2:
1. I wonder whether it is possible to miss the real bug, when the programmer changes the slicing criteria, and the slice Navigator recompute a new slice during debugging?

Yes, it is a general problem of slicing that when developers mistakenly identify an erroneous value as correct, the fault can be removed from the program. This is out of scope of this work.

However, specifically with the Slice Navigator, once developers realize the entire state in the slice is correct, they have to rewind their debug session. Here the requested undo-button could be used until erroneous state can be found again.

2. (line 172): The author should present more details about the slice algorithm for the readers for understanding it easily.

The aim of section 3 was to describe the tool from the user’s perspective. We moved the part about the slicer down to section 4.1, where we also added more details about the algorithm.

The proper way is to put the algorithm described when the first time mentioned, rather than at the end of this paper.

We agree. This was caused by LaTeX and we could not easily change it, but this is not the final layout for publication.

3. I think, it will be better to present a formal presentation of the slice which is very useful for developers understanding. (in Section 3). The formal description of events in the slice is also helpful.

We are not sure if we understand this feedback correctly. Typically, visualizing the full slice is not helpful for developers because it is simply too large to be manually analyzed in any meaningful way.

4. (Fig 6) In this paper, the author only gave three bugs debugging scenario, why not conduct more experiments to support the findings by the author?

We agree that having more bugs in the study would be desirable, but we found it difficult to find bugs that can be located in a short time with no prior knowledge of the code, so we focused on getting a good number of measurements for each bug instead. All bugs are real world bugs and represent different classes of bugs.

5. (SubSection 5.2) I suggested the authors to group the participants involved in the user survey and give a detailed description of the empirical study with a list. Because the reader is difficult to identify the time cost and the views of participants accordingly.

We added more details about the two subgroups (students and professionals) in section 5.2.5. We could add more details about each participant, but there was no correlation between time needed for a task and other observations or experience reports, so we don't see a value in listing those details.