**Interview Protocol**

**Introduction**

We will be conducting semi-structured interviews to explore developers' debugging practices and gather feedback on the potential integration of dual-version slicing techniques into existing IDEs. These techniques aim to reduce the code developers need to inspect when debugging, offering enhancements like side-by-side version comparisons, variable inspection, and statement categorization.

The interviews will take approximately 30 minutes to 1 hour, conducted via Zoom, and will be audio-recorded with participants' consent.

**Objective**

To understand the potential usefulness of dual-version slicing, how developers interact with different types of statements, and how they envision integrating these techniques into their daily debugging environments.

**Interview Structure**

**Part 1: Introduction and Debugging Practices**

* **Goal:** Understand participants' current debugging practices.
* **Questions:**
  1. **Q1:** Regression bugs occur when changes cause previously functioning features to break. How do you debug in these situations (e.g., printing, breakpoints)?
  2. **Q2:** What is the typical number of files or statements you inspect during a test failure?

**Part 2: Introducing Slicing Techniques**

* **Goal:** Introduce slicing as a solution to reduce the code base during debugging.
* **Introduction:** We have an approach that reduces 35k statements to about 500 relevant ones. That’s over 95% code reduction.
* **Q3:** Do you think this kind of code reduction would be useful?

**Part 3: Integrating Slices into IDEs**

* **Goal:** Explore how developers envision using slices in their IDEs.
* **Introduction:** Not all statements in a slice are equally important. Some are changed statements, others propagate values.
* **Q4:** Slice lines can appear in different files and methods. How would you envision integrating these into your IDE? Any ideas on how to make it more useful for your daily debugging?

**Part 4: Visualizing Statements**

* **Goal:** Discuss the visual representation of important vs. less important statements in the IDE.
* **Introduction:** We propose highlighting essential statements (executed changes), graying out secondary ones, and folding non-important statements with the flexibility to unfold them. We could even cross out unexecuted statements.
* **Q4 (Cont.):** What do you think about this modified editor view with these features?

**Part 5: Debugging Using Statement Categories**

* **Goal:** Understand how developers debug when statements are categorized as essential, secondary, or non-important.
* **Introduction:** We assume you would skip non-important statements and see only important variables in the variable window. You can choose to debug essentials, secondary, or all.
* **Q5:** How would you debug using these categories of statements?
* **Q5 (Cont.):** For instance, in line 19, what statement would you want to see next: line 20 or 26? How would you prefer this to be visualized?

**Part 6: Side-by-Side Editor View**

* **Goal:** Gather feedback on the side-by-side view of two program versions.
* **Introduction:** We synchronize steps between two executions in the side-by-side editor.
* **Q6:** Do you find the side-by-side editor view useful? If not, how would you like to see information from the old passing version?

**Part 7: Usefulness of Textual Explanations**

* **Goal:** Understand whether participants find textual explanations helpful.
* **Introduction:** Imagine hovering text that explains a block of less important statements.
* **Q7:** Do you find textual explanations like this useful? Any other ideas for presenting information?

**Conclusion**

* Thank the participant for their time.
* Inform them that the feedback will contribute to improving slicing techniques and integrating them into IDEs.
* Reassure them of confidentiality and remind them that transcripts will be anonymized for further analysis