**Semi-Structure Interview Protocol**

**Part 1: Introductions**

* **Goal:** Introduce the team and the study to participants. Explain that the objective is to explore 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 workflows.
* **Study Procedure:** Clarify that the interview is structured as an open discussion rather than a rigid question-and-answer session. Participants will be asked a series of questions designed to explore their experiences and thoughts, but they are encouraged to share their perspectives freely. The flow of the interview may evolve based on their responses, allowing for a more natural conversation. Participants are free to skip any questions they do not wish to answer without needing to provide a reason. The interview is expected to take up to one hour, though the length may vary depending on the depth of the discussion and participants’ availability.
* **Privacy and Data Storage Policies:** Inform participants that their confidentiality will be respected. Participant identities will remain fully anonymized, and only aggregated results will be reported. The study results will be published in an academic paper and made openly available to the community. All data collected will be stored securely in encrypted, password-protected storage.
* **Obtain Verbal Consent:** Ask participants for their verbal consent before starting the interview.

**Part 2: Participant Background**

* **Goal**: Collect participants' professional background, experience, and current development roles.
* **Questions**:

**Q1**: Please tell us about your current role and how long you've been working in software development?

**Q2**: What programming languages and development environments do you typically work with?

**Q3**: Could you describe the types of projects you usually work on (e.g., web development, systems programming, mobile apps)?

**Part 3: Debugging Practices**

* **Goal**: Understand participants' current debugging practices and their overall experience with regression bugs.
* **Questions**:

**Q4**: How often do you encounter bugs during your development work? What types of bugs are the most challenging to resolve?

**Q5**: What is the typical number of files or statements you inspect during a test failure? How do you manage debugging when the scope is large?

**Q6:** What methods or tools do you typically use, such as printing, breakpoints, or debuggers?

**Introduce regression bugs**: Explain that regression bugs occur when changes to software, e.g., to add new features or fix security vulnerabilities, unintentionally break existing functionality (Think “it worked yesterday but does not work today”).

**Q7**: What is your experience with debugging regression bugs? could you tell us about a specific instance that was particularly difficult or interesting?

**Part 4: Introducing Slicing Techniques**

* **Goal**: Introduce slicing as a solution to reduce the code base during debugging.
* **Introduction**: Slicing reduces the scope of code developers need to inspect by around 90%, e.g., for the projects we analyzed with 35k statements, slicing may reduce it to 500 relevant ones. That’s over 95% code reduction.
* **Questions**:

**Q8**: Do you think this kind of code reduction would be useful in your workflow? How do you usually identify which parts of the codebase are relevant when debugging?

**Q9**: How would you envision integrating slice statements into your IDE? How would this fit into your current debugging process?

**Part 5: Visualizing Statements**

* **Goal**: Discuss the visual representation of essential, secondary, and irrelevant statements in the IDE.
* **Introduction**: We propose highlighting essential statements (e.g., executed changes), graying out secondary ones, and folding irrelevant statements with the flexibility to unfold them. We could even cross out unexecuted statements.
* **Questions**:

**Q10**: What do you think about this modified editor view with these features? How would this change your debugging workflow?

**Q11**: How important is the visual organization of code during debugging? Would you find features like folding or highlighting particularly useful in complex codebases?

**Part 6: Debugging Using Statement Categories**

* **Goal**: Understand how developers debug when statements are categorized as primary, secondary, or irrelevant.
* **Introduction**: We assume you would skip irrelevant statements and see only important variables in the variable window. You can choose to debug essentials, secondary, or all.
* **Questions**:

**Q12**: How would you debug using these categories of statements? For example, in line 19, what statement would you want to see next: line 20 or 26? How would you prefer this to be visualized?

**Part 7: Side-by-Side Editor View**

* **Goal**: Collect feedback on the side-by-side view of two program versions.
* **Introduction**: We synchronize steps between two executions in the side-by-side editor.
* **Questions**:

**Q13**: Do you find the side-by-side editor view useful during regression debugging? If not, how would you like to see information from the old passing version?

**Q14**: Have you used tools that provide version comparisons before? If so, how did they support or hinder your debugging efforts?

**Part 8: Usefulness of Textual Explanations**

* **Goal**: Understand whether participants find textual explanations helpful.
* **Introduction**: Imagine hovering over text that explains a block of less important statements.
* **Questions**:

**Q15**: Do you find textual explanations like this useful? How would you improve them? Are there other ways you'd like to see information presented in the IDE?

**Conclusion**

* Thank the participant for their time.
* Inform them that their 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.