**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 features like side-by-side version comparisons, variable inspection, and statement categorization.

The interviews will take approximately 30 minutes to 1 hour, conducted via Zoom.

**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 workflows.

**Interview Structure**

**Part 1: Introduction and Background**

* **Goal**: Understand participants' professional background, experience, and current development roles.
* **Questions**:
  1. **Q1**: Can you start by telling us about your current role and how long you've been working in software development?
  2. **Q2**: What programming languages and development environments do you typically work with?
  3. **Q3**: Could you tell us about the types of projects you usually work on (e.g., web development, systems programming, mobile apps)?

**Part 2: Debugging Practices**

* **Goal**: Understand participants' current debugging practices and their overall experience with regression bugs.
* **Questions**:
  1. **Q4**: How often do you encounter bugs during your development work? What types of bugs are the most challenging to resolve?
  2. **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?
  3. **Q6:** What methods or tools do you typically use, such as printing, breakpoints, or debuggers?
  4. **Q7**: Do you often collaborate with other developers during the debugging process? If so, how do you communicate findings or work together on solving bugs?
  5. **Q8**: Regression bugs occur when changes cause previously functioning features to break. Can you tell us about your experience with debugging regression bugs? could you tell us about a specific instance that was particularly difficult or interesting?

**Part 3: 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.
* **Questions**:
  1. **Q9**: Do you think this kind of code reduction would be useful in your workflow? Have you used any other techniques or tools to minimize the code you inspect? How did they compare?
  2. **Q10**: How do you usually identify which parts of the codebase are relevant when debugging? Would a tool that automatically narrows down code be helpful?

**Part 4: 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.
* **Questions**:
  1. **Q11**: Slice lines can appear in different files and methods. How would you envision integrating these into your IDE? How would this fit into your current debugging process? Any specific features you’d want for better usability in your daily debugging activities?
  2. **Q12**: Have you previously used debugging tools that offer code reduction or slicing? How did they impact your productivity?

**Part 5: 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.
* **Questions**:
  1. **Q13**: What do you think about this modified editor view with these features? How would this change your debugging workflow?
  2. **Q14**: 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 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.
* **Questions**:
  1. **Q15**: How would you debug using these categories of statements? How would you choose which statements to focus on?
  2. **Q16**: 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**: 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.
* **Questions**:
  1. **Q17**: Do you find the side-by-side editor view useful when comparing different versions of code? If not, how would you like to see information from the old passing version?
  2. **Q18**: 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**:
  1. **Q19**: 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.