Glenn Hanawalt (grh95) and Madeline Wessels (mwes)
CSE 490 E1
Assignment 3
1/10/17

Proposed Case Study of the Index Checker Tool

Project:

The focus of this project is to examine the Index Checker, an existing tool developed in Java [1]. This tool, according to core developer Joseph Santino, is a “type checker that issues warnings about array, list, and string accesses that are potentially unsafe.” Establishing the soundness of this tool will be an undertaking requiring tests on dummy projects and eventually a case study on larger scale repositories.

Motivation:

While solving known faults is relatively easy, finding faults is challenging and proving their absence is more difficult still. IndexOutOfBoundsExceptions (IOOBEs) occur frequently, and preventing these errors in development can avoid vulnerabilities and expensive fixes. Various approaches to detecting such errors exist. Dynamic analysis tools such as Valgrind [2] and Purify [3] offer a run-based error detection, spitting flags and warnings at the developer to alert attention to areas of weakness. Heuristic tools such as ESPX [4] and FindBugs [5] use static analysis to find bugs: the former infers specifications while the latter searches for code patterns.

Type systems tend to be easy for programmers to use and produce understandable errors. Preventing IOOBEs with strong type systems is an effective method for improving a system’s resilience. The Index Checker uses an annotation-based type system to turn common runtime errors into compile-time checks. This is preferable to a runtime exception because it guarantees that programs that pass the check can never crash from IOOBEs.

Goals:

The Index Checker has great potential value, but its effectiveness and usability have not been established. This project seeks to assess the tool’s accuracy, bug detection, and usability to determine whether it can serve as a valuable tool for improving software quality in large-scale projects.

Approach:

The Index Checker will be assessed in five stages. First, the team will become familiar with the Index Checker’s documentation, usage, and source code. Second, analysis of the tool will begin on a small scale, using toy cases to verify performance. Prevented exceptions, unprevented exceptions, and false alarms will be described and documented. Third, assessment will move to a large-scale case study on an existing, real-world codebase to observe how the tool scales to larger programs. Fourth, the team will evaluate the collected data and their experiences, reporting on the tool’s current state, evaluated for its usability and soundness. Finally, The results and reflection acquired from the study will then be used to improve upon the Index

Checker, if necessary. Depending on what changes have been made since the publication first introduced the tool, it may be worthwhile to implement the extensions that were mentioned by the author, such as addressing arithmetic on array index offsets. Figure 1 shows the workflow proposed above.

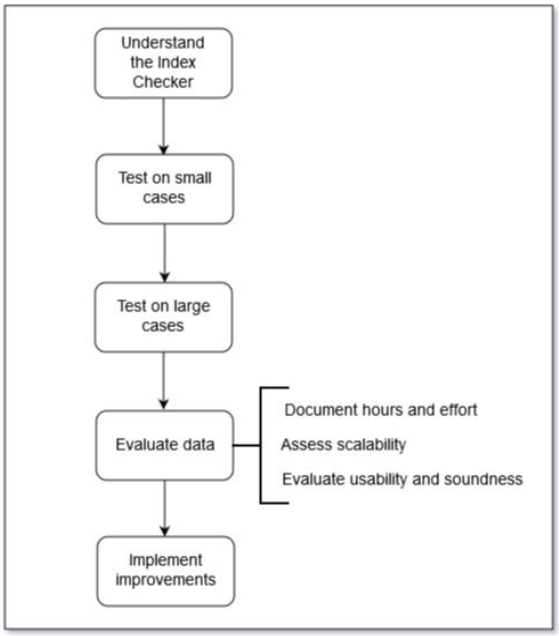

Figure 1: Proposed workflow diagram.

Challenges and Risks:

Establishing a test suite for the checker tool will be the most significant challenge for this project. The development challenge can be broken down into smaller, achievable tasks. Before development, the checker tool's API must be understood. Establishing dummy code to run the checker against provides insight into the tool. After this simplified use-case, the internals of the checker tool must be plumbed. Code coverage reports allow determination of the extent of testing within Index Checker's development. Expanding on these tests to cover more methods is vital towards development of a richer Index Checker.

Another challenge is evaluating the tool from a programmer's perspective. Usability is a major concern for any tool, but it's also subjective and hard to quantify. In order for this study to be useful, the team must evaluate the difficulty of adding the Index Checker annotations to a real codebase in addition to the number of bugs that it prevents or reveals. Reporting the programmer effort necessary to use the tool would allow, say, a project manager to make a more informed choice of whether or not to use the tool. Therefore, the team will carefully document the time necessary to learn the tool and apply it to real codebases, using time as a quantifiable metric for usability.

References

1. Santino, Joseph. "Enforcing correct array indexes with a type system." *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016* (2016): 3. Pag.
2. N. Nethercote and J. Seward. "Valgrind: A framework for heavyweight dynamic binary insrumentation." In *PLDI 2007, Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation*, pages 89–100, San Diego, CA, USA, June 11–13, 2007.
3. R. Hastings and B. Joyce. "Purify: A tool for detecting memory leaks and access errors in C and C++ programs." In *Proceedings of the Winter 1992 USENIX Conference,* pages 125–138, San Francisco, CA, USA, January 20–24, 1992
4. B. Hackett, M. Das, D. Wang, and Z. Yang. "Modular checking for buffer overflows in the large." In *ICSE'06, Proceedings of the 28th International Conference on Software Engineering*, pages 232–241, Shanghai, China, May 24–26, 2006.
5. D. Hovemeyer and W. Pugh. "Finding bugs is easy." In *Companion to Object-Oriented Programming Systems, Languages, and Applications (OOPSLA 2004)*, pages 132–136, Vancouver, BC, Canada, October 26–28, 2004.