Example Project Proposal: Swift

Student Name(s): Kyle Dewey
Proposed System Under Test (SUT): swift
Link to SUT Source Code: https://github.com/apple/swift
SUT Size: 262,362 Lines of Code

SUT Description
The compiler and runtime of the Swift programming language (https://developer.apple.com/swift/), developed by Apple. Swift is a statically-typed language featuring generics and higher-order functions. I did not make the SUT and am unfamiliar with it. I'm expecting a bonus as a result.

State of SUT
Swift is relatively mature (publicly existed since 2014), and is actively developed (over 76 thousand commits, including those from an hour ago). The language is being incrementally updated. There is no formal specification, but there is extensive documentation (https://swift.org/documentation/).

Attributes
- **Interactive**: Swift allows programmers to incrementally write code and see code results.
- **Modern:** Swift makes use of the latest programming language features.
- **Safe:** Swift's design prevents many common exploitable programming errors.
- **Fast:** Swift code runs quickly.

Components
- **REPL (Read/Eval/Print/Loop):** Allows users to incrementally type Swift code in a command-line interface and see the results.
- **Typechecker:** Ensures that Swift programs are well-typed.
- **Code Generator:** Compiles Swift code to machine code, with LLVM (https://llvm.org/) bitcode as an intermediate language.
- **ARC (Automatic Reference Counting)**: Automatically deallocates memory that is no longer needed at runtime.

Capabilities:
- **Typechecker** is **Modern**: The typechecker can check if generics are used properly.
- **Typechecker** is **Modern**: The typechecker can check if higher-order functions are used properly.

Unit Tested Capabilities(s):
Unit tests will be written for the typechecker which involve generic programs and programs with higher-order functions. Swift is a complex language with a correspondingly large typechecker.

Automatically Tested Capabilities(s):
I plan to test the typechecker with grammar-based approaches. Specifically, I plan to automatically generate well-typed programs involving generics and higher-order functions. I'll know a bug is found if the compiler rejects any of these programs, as they should all be well-typed.