Example Project Proposal:
Heavy Advanced Testing

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.

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:** Compiler Swift code to machine code.
- **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.

Capabilities Count:

	Interactive	Modern	Safe	Fast
REPL	0	0	0	0
Typechecker	0	2	0	0
Code Generator	0	0	0	0
ARC	0	0	0	0

Basic Testing:
For each one of the listed capabilities above, I plan to write unit/integration tests as appropriate to test each one. Swift is already heavily tested, so this should not require dramatic modification to its codebase.

Advanced V&V:
I want to use grammar-based techniques to generate Swift programs with which to test the Swift compiler. My plan is to design a grammar that describes a subset of well-typed Swift programs, and then use that grammar to generate well-typed Swift programs. If the Swift compiler fails to accept any of these programs, it indicates a bug in Swift's typechecker.

Additionally, I plan to use test coverage and mutation analysis to gauge the effectiveness of this automated testing, relative to Swift's existing test suite. My goal is to have the advanced testing approach + Swift's existing test suite outperform Swift's existing test suite alone. Mutation analysis is made possible with mull (https://github.com/mull-project/mull), which performs mutation testing on LLVM bitcode.