**TEAM: 4 SSDynamics**

**Overview:** The main purpose of the “Technical Demos” is to very clearly communicate the extent to which the team has identified key challenges in the project, and has proven solutions to those challenges. Grading is based on how complete/accurate the list of challenges is, and how convincingly and completely the given demos cover the given challenges.

This template is fleshed out by the team, approved by CS mentor, and brought to demo as a grading sheet.

### Risky technical challenges

Based on our requirements acquisition work and current understanding of the problem and envisioned solution, the following are the key technical challenges that we will need to overcome in implementing our solution:

**C1: Random Test Key.** The program must be able to generate randomized test keys to create random traversal through the state graph for a given command. The key must also be saved so the tester can re-run the same test to recreate any issues that occur. It must also be clear to the user that a different key is being used

**C2: Ability to parse TLA+ Constants w/ error handling.** The program must be able to parse TLA+ constants from a json file to be used as constants with each TLA+ file instead of initializing them inside the TLA+ file and in the NVMe command.

**C3: Logging Output.** The program must be able to output a rolling log, which can have the length of the log changed. The program must also be able to categorize errors, and output all errors at the end of the log.

**C4: NVMe CLI Command Calls.** The NVMe CLI must be able to communicate with the rest of the Python code, this is a one way communication between the Python executor and the NVMe CLI library. There is a dedicated set of commands that we can execute through NVMe CLI, and covered by the NVMe specification.

### Challenges covered by demos:

In this section, we outline the demonstrations we have prepared, and exactly which of the challenge(s) each one of them proves a solution to.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**( 24/25) Demonstration 1: Run the simulator and demonstrate it sampling different seeds**

Challenges addressed: Random Test key

Flight Plan: Step by step overview of demo

1. Double check that run\_test\_no\_nvme is active in interface.py
2. run ./nvme\_gentest and run SSD\_NAU\_TEST.tla
3. analyze the seeding and randomness

Evaluation:

* Convincingly demo’d each of listed challenges?

Yes.

* Other evaluative comments:

Asked the team to demonstrate re-running the program with the same seed to see the output and asked them about reproducibility, which they convincingly answered. Note: there is a bug in the output of different states (they do not enumerate properly).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**( 25/25) Demonstration 2: Creates a rolling log output with errors**

Challenges addressed: Logging

Flight Plan: Step by step overview of demo

1. Demonstrate a run of the ./nvme\_gentest with a tla+ file that shows no errors
2. Demonstrate a run of the ./nvme\_gentest with a tla+ file with errors
3. Then show a run with a tla+ file that has a different line length, so first with a line length of 100, and a line length of 1000

Evaluation:

* Convincingly demo’d each of listed challenges?

Yes

* Other evaluative comments:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**( 24/25) Demonstration 3: TLA+ constants with error handling**

Challenges addressed: TLA+ constants

Flight Plan: Step by step overview of demo

1. Demonstrate the parsing of the constants from a json file to be used in a tla+ file by running ./nvme\_gentest without constant data in the command line.
2. Demonstrate error handling by removing a required constant from the json file, and running ./nvme\_gentest

Evaluation:

* Convincingly demo’d each of listed challenges?

Yes

* Other evaluative comments:

We also talked about what error handling looks like when the constant exists but has a missing value. Would like to see the team test them with invalid data types in the constants and verify that error handling holds for those cases as well.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**( 25/25) Demonstration 4: NVMe CLI Command Calls**

Challenges addressed: Calling direct NVMe CLI control commands from the Python script

Flight Plan: Step by step overview of demo

1. Use the executor.py file to directly demonstrate the use of the NVMe CLI.

OR

1. Run the TLA+ file through the generative testing platform as it currently exists.

Evaluation:

* Convincingly demo’d each of listed challenges?

Yes

* Other evaluative comments:

### Other challenges recognized but not addressed by demo:

Two way communication between the TLA+ and NVMe CLI interface isn’t quite possible, however, we can communicate two way from the NVMe CLI to the Python TLA handler, which is something that we are still actively working on. Another challenge not addressed by the demo is concurrent handling of events, the program currently only works in a concurrent fashion, so each command is called and logged one after the other.

**Other Notes:**

We discussed what this product’s use looks like from an engineer’s perspective and also checked in on completion level and the team’s confidence in delivering their final product to their client, which seems to be on track.