My Experience:

Testing dominion at first seemed like a daunting task. Having never played the game before much of the code at first glance was rather confusing. The cardEffect function seemed pretty indecipherable, and it was tough to figure out just how the game was keeping track of coins, buys, and actions. After getting very familiar with the game (my roommate and I now have an unhealthy obsession and have spent way too much money on its expansions,) all the moving parts became much clearer.

This manifests itself directly in the quality of my tests over time. My first unit tests we're honestly very simple. For instance, my one test just checked to make sure that there were 12 provinces at the start of the game. My card tests basically just checked to make sure that if you played the card it wouldn't return an error. As I got more familiar with the code I was able to write better tests. My random tests not only had more coverage but were much more comprehensive than any of my unit or card tests as they tested for more specific criteria. For instance, testing to make sure that if adventurer was played you not only had more cards but that you have the correct hand count and the correct coin count.

Testing all of dominion, however, was pretty frustrating. Despite any tweaks I made, dominion.c was still buggy. My random tester would get caught in infinite loops constantly. I discovered later it was because the number of player buys and actions weren't being subtracted properly. To find this bug took me hours, having my test print out every action played to try and narrow down what card was throwing me for a loop. Agan's divide and conquer rule was used to good effect, though it would probably have been more beneficial if I took his first rule a little bit more seriously and spent more time just studying the code.

Coverage:

My coverage for my unit and card tests was 18.76% of 565 lines. Considering I was testing only small portions of the code this isn't an awful number. With the random tests that we're written later I made sure, using the gcov functionality that breaks down the coverage of every function, that my cardEffect replacements were being tested with 100% coverage. After running dominion a number of times, my full game dominion tester could achieve 61.4% coverage. However, there is still a bug that results in an infinite loop, different from the one mentioned above, that probably hurt my coverage quite a bit. I put in a hack to ensure that the loop would break after an unreasonable amount of iterations, however this means that the whole turn wasn't played out correctly possibly hindering my coverage.

Classmate 1:

Classmate 1's dominion.c was actually very impressive. He or she decided to completely redo the cardEffect function, separating each card into its own function in cardEffects.c. This made the code not only easy to understand, but also easy to test (though it took me a second to get my tests to compile correctly.) It also provided much stability to the code base.

Coverage:

1. **Unit Tests:** The unit tests had 100% of their lines executed for each card and forty percent coverage of the cardEffects.c.
2. **Card Tests:** The card tests tested four cards, ambassador, baron, salvager and council room and also covered 100%. All of the unit and card tests passed without finding any bugs.
3. **Random Tests:** The random tests provided surprisingly good coverage of the code. For each card tested, 100% of the lines were executed. Overall the test executed on average 40%, the highest being 47.29%. All tests passed and no bugs were found with the random testers which leads me to believe that the code is reliable, definitely moreso than the base dominion.c we were given to work with.
4. **TestDominion:** Running my dominion tester on my classmate's code proved to be a little tricky, mostly getting the makefile to compile my tester and doing a little bit of refactoring in order to play all the cards in my classmates cardEffects.c. I however didn't find any bugs, which leads me to believe my tester might be lacking in some areas. I trust the code more than I trust my test, which should never be the case.

Classmate 2:

Classmate 2's dominion.c was much closer to mine which made the testing much easier to accomplish. Overall classmate 2's was very similar to mine, with my testdominion reaching over 60% on both of our code. If his code is only as reliable as mine is than at least one of its legs is a little wobbly.

Coverage:

1. **Unit Tests:** The unit tests all passed and averaged about 17% coverage of dominion.c
2. **Card Tests:** The card tests all passed as well but after running they averaged a much higher 25% coverage of dominion.c. This goes to show how much of the code is tangled as running these functions was able to cover 25% of the code base. Having these functions pass their tests proves that a fairly large portion of the code is reliable.
3. **Random Tests:** The random test for both council room and smithy were both very effective. Both of them easily hitting 100% coverage of the test and the card itself. These are the easiest

cards to test, however, so having 100% coverage of these cards doesn't say a ton for the reliability of the overall code. The random test for adventurer, however, did not pass. I attempted to look into the bug and it seems to be after the card is played but before the players cards are discarded. I wasn't able to find the bug to fix it.

4. **TestDominion:** Classmate 2's testdominion was impressive, covering up to 78% of the code base. My testdominion, when run on his/her code only achieved 64% code coverage. I also didn't find any bugs, which was expected as classmate 2's code coverage passed and covered more code. Running diffdominion on our testers was a failure, as not once were both outcomes similar. The difference in coverage isn't enough to say that either is necessarily more reliable than the other, due to the randomization of dominion.