Milestone 3: Testing Experiences

# Our Project: Dumb Chess

## Testing Methods Picked

### Blake

I was in charge of the individual piece classes, so I focused mainly on unit tests. I used equivalence class testing within the unit tests to make sure to cover all different equivalences cases. I made sure to test each piece’s movement based on different positions on the board, such as on an edge, with pieces in the way, and wide open in the middle of the board.

The whole team also focused on doing code reviews for all new code. This helped catch bugs at an early stage, and also to find issues that weren’t yet bugs, but would have caused problems in the future. It was also helpful for all of us to be familiar with everybody else’s code, and how to integrate everything.

### Hoyoung

With dumb chess. I was in charge of the chessboard. My goal was to make sure the chessboard existed with boundaries and also contained functions for chess board to send and receive data for movement of pieces and adjusting for piece movement, setting the piece to go when needed.

With tests, I used the testing method of requirement testing and boundary testing. Some boundary checks were able to be done by the chessboard as chessboard had limitations of 8 by 8. For requirement testing, the other functions that relied on the chessboard needed specific, so I utilized type checking and error guessing.

### Jaden

I was in charge of making sure all the application’s different components (chess board, chess game, and pieces) interacted with one another as they should in traditional chess. I was also in charge of writing the main driver for the chess program and implementing the program’s error handling code and user input. I used the program specification to perform the requirement tests needed to verify the basic program functionality including loading the chessboard, managing player turns, prompting players for each turn, pieces moving to the correct spot on the chess board after a valid input was entered, making sure the chessboard gets updated for each turn, and handling any user input formatting errors. Because I was testing how each component interacted with one another, integration testing played a huge role in testing the program in its final build and running through multiple playtests. I also emphasized usability tests which make entering commands less strict in their format while still conforming to the requirements laid out in the specification.

### Joseph

When it comes to games, there is perhaps no testing method more important than playtesting. Playtesting will find the bugs that most affect the player experience, while bugs that go unnoticed by playtesting are often unimportant enough that they don’t need to be fixed.

Since we can review the code for the game, another method I placed great emphasis on was doing code reviews. These can catch a lot of problems early on and encourage a more consistent code structure. It also gives team members opportunities to discuss coding style with each other.

## Additional Tests

### Blake

Beyond the unit tests that I wrote and ran, I also did some playtests to make sure all the parts of the software worked together well. I used some error guessing in my playtesting as I played the game. I tried doing illegal moves, capturing enemy pieces, capturing my own pieces, and more. I found from this testing that everything worked well.

### Hoyoung

Some additional test I ran was, I played the game few times through and check to see if errors were coming out right where they needed to be and checked to see if anything else was missing.

### Jaden

Some additional tests I performed involved selecting friendly pieces and attempting to move them to legal spots on the board where enemy pieces of the same type could move to to check that the program was properly taking into account which team the selected pieces were part of. I also entered legal moves and reversed the order of the spots to check if the program was taking into account the order and formatting of the user commands. Fortunately, the program handled both these scenarios as expected from the specification and no extra bugs were found. I also had a friend try playing against me in a game of chess and took note of any confusion they would have had during the playthrough. Thankfully, the program seemed pretty straightforward in their view and no additional usability bugs were found from this.

### Joseph

I didn’t do any tests beyond our test plan except for a bunch of quick solo playtests. These were easy ways for me to ensure that a feature was working in the context of the rest of the game before I moved on from it.

## Test Effectiveness

### Blake

I found that the testing methods I employed proved to be quite helpful. The unit tests were able to show with confidence that the individual piece classes work well, and can’t be moved incorrectly. It showed that it won’t allow for pieces to capture other pieces of the same team, nor move off the board or anything else.

### Hoyoung

I would say, the error guessing, type checking and boundary testing have been very effective. The type-checking assisted in finding possible errors between working with team mates, and error guessing assisted with negative testing. Boundary tests seemed great for checking a bit more thorough on each edges, successfully continuing when it should be and failing on where it should fail.

### Jaden

I would say that the requirement testing was very effective. The testing against the program specification has highlighted the importance of having a precise and accurate vision for a software application, which I now understand can make the testing process much easier when done carefully. The encapsulation of the program in a chess game class also has made testing specific features easier, especially when certain features rely on specific functions from multiple program components. Because of this, integration testing became significantly easier and many potential bugs would have been caught if we hadn’t used the specific architecture we had used.

### Joseph

Unfortunately, team scheduling issues prevented most of the playtesting and code reviews from being done in time for shipping milestone 1. Still, playtesting was very effective at uncovering bugs and usability issues. Code reviews also helped to significantly reduce the potential for weird bugs that could be difficult to pin down with playtesting.

## When to Stop Testing

### Blake

I established specific stopping criteria for, and I stopped testing upon meeting those conditions. These criteria entailed exhaustively verifying the equivalence classes for piece movement, continuing until all movement patterns I could think of were tested and potential errors had been accounted for.

### Hoyoung

I felt that at certain point, the tests became redundant as it seemed the tests produced the same results. With edge case and type checking, i was able to ensure that other functions were able to use as the program was intended to be used and not to exceed any boundaries.

### Jaden

I stopped testing once I had observed the expected program functionality from different use cases. My first set of use cases were designed to “test-to-pass", which consisted of only entering legal moves and selecting appropriate pieces based on which player’s turn it was. My next set of test cases consisted of testing valid inputs which would result in the program throwing an error. This includes selecting valid pieces but attempting to move them to valid spots on the chess board that were not within that piece’s range of legal moves. My final set of use cases consisted of entering illegal inputs and “testing-to-fail". I stopped entirely once I was able to test a significant number of inputs in each use case and observe the expected output from each test.

### Joseph

I decided that it was time to stop testing when every feature had been shown to work in a playtest and all the major usability issues were ironed out. Additional testing may have found more bugs, but time constraints were such a significant limiting factor that many of the tests weren’t done in time for milestone 1. There was little reason to do any more tests at that point.

# Another Team’s Project: Telephone Switching

## Testing Methods Picked

### Blake

I decided to come up with boundary tests and edge cases for testing this software. There are lots of areas within this software that can be tested with boundary testing, including multiple facets of the phones.txt input file, user input, and different use cases of calling, like specific conference calling.

### Hoyoung

I came up with tests for this mostly using requirement testing and error guessing tests. Lots of requirement tests overlapped with others as we had very similar ideas on what should be tested. And project outline gave a good boundary to show where the program should be failing and erroring out.

### Jaden

I mainly used error guessing and requirement testing techniques. Thankfully, the program specification made it very easy to identify certain boundary conditions that would need to be tested. Lots of the tests we distributed amongst ourselves overlapped, so on top of performing the tests I was initially assigned, I attempted to identify additional usability and requirement bugs by testing the program’s behavior and comparing it to the specification to see if any ambiguity in the specification could be found. There were only two very small and trivial bugs which didn’t affect the core behavior of the application, and the rest of the program ran pretty well.

### Joseph

The detailed requirements made the idea of doing requirements testing an obvious one, so I happily went with that.

## Additional Tests

### Blake

After performing my assigned tests from the test plan, I performed a few more tests. This was because after doing my tests, and had more thoughts of areas that should be tested. I only did a couple more tests, and they all passed.

### Hoyoung

I performed tests that were assigned to me and I tried to use the program the way i felt like it would be used, such as trying to put every user on a conference call and seeing if there was an issue with individual person and functionality of the calling program. 20 user isn’t significate and the program seemed to work and handle well with the total load that it was able to handle.

### Jaden

Once I finished my initial tests, as well as the additional checks for ambiguity in the specification, I decided the rest of the program behaved as expected given that all other test cases would have been covered by my fellow team members.

### Joseph

I did not do any testing beyond our test plan.

## Test Effectiveness

### Blake

I think that the boundary tests I came up with were effective. Most of them passed, showing that the developers handled many edges and boundaries effectively. One of them did fail, which will now be insightful for the developers to go back and cover a boundary they hadn’t thought of before.

### Hoyoung

I felt that the tests were very effective as we were able to find some bugs that i felt it broke the program. Such as finding bugs that could not handle errors and crashing causing the program to be unusable. My personal tests seemed not very effective as the users were limited to very small amount and it was easy to assume that most devices and programs should handle the small amount of users.

### Jaden

Many of the boundary tests identified from the specification passed. However, there were two specific edge cases which the original developers did not take into consideration. Unfortunately, the program would just crash whenever this particular event occurred. This proved the requirement testing to be effective in helping identify bugs in the software, and given this insight, future developers could easily fix it given what we had found.

### Joseph

I believe that the requirements testing was very effective at ensuring that the program did what it was supposed to do. It glossed over some usability issues, but we used other testing methods to cover that area.

## When to Stop Testing

### Blake

I used my assigned tests to help me decide when to stop testing. As mentioned before, I ended up running a couple more tests beyond my assigned ones, and then stopped once I felt we had covered all areas defined in the program spec.

### Hoyoung

I stopped testing after i did my assigned tests and tried to run a few of my own to see if i can find a bug in the load capacity. As mentioned before, my personal tests were not very effective and small user seemed a bit useless to test the functional load limit.

### Jaden

After I finished my assigned tests, I decided to try making a few error guesses to see how the program would run in a few less common use case scenarios. I only found a couple minor usability bugs which did not affect the base functionality of the program in any way. It was at this point when I decided to stop testing since all other use cases were covered by our other tests.

### Joseph

I decided to stop testing upon completing all the tests that I had assigned to myself.