Sarah Schuster-Johnson
Oracles

I chose to use multiple compiler versions, as well as running a debugger, “lldb“ from the command line. This tool was very handy and helpful because it allowed me to step through my code, just like xcode, but with different compilers.
I used the -O0, -01 and -02 versions of GCC to compile my code. Compilers are written by people, to check the standards of the language, your implementations, and if everything follows suit. Therefore, different compilers might find errors another one might not. The compiler is automated, and makes sure there are no errors in it’s implementation, variables are assigned, arrays are allocated correctly, memory is allocated correctly, etc. In the past, I have compiled my code in xcode, then went to compile with GCC in the command line, and it found small errors. These errors were not detected in xcode, and allowed my code to execute without alerting me to an issue.

I chose to use Assertions. I used this method to assert if all my conditions were met and followed as my code ran, then my assertions without a doubt should be true. This was a nice method because it can catch errors that perhaps would result with the intended result in one occurrence, but later down the line might cause a serious error. Using this method during the final testing period was helpful, however I believe this would have been a much more helpful to me if I had used it more often I was developing my code to check for errors. I could see assertions being a powerful tool when writing large code , when errors might be trickier to find using step throughs, or simply relying on the output. Assertions forced me to think a bit more precisely as I looked over my code. I could see how this tool could help shape a more efficient, clean coder. I used the assertions to ensure my data variables were indeed filled in correctly. I used assertions to clarify if a shape ended up classifying in a particular way, it in fact could could not classify as another shape. I used the assertions mostly to ensure all data was filled in correctly, and that I didn’t not incorrectly classify a shape.

There were two final testing methods I implemented that helped me significantly with my testing. I wrote my data points out to a file. I then plotted each data file, using gnuplot, my shapes in order to undergo a visual inspection of my shapes. This helped me find bugs and errors within the confines of human error. It was very helpful, and also very interesting, to see a visual representation of my coordinates (this was especially helpful when implementing my random number generator). There is a bash script and gnuplot script included in my repository if you are interested in how I implemented this.

My final method for testing was to overwhelm my classifier with data. I ran my program giving it 1,000 files, and each file containing 2,000 data lines. This became the most helpful for finding small, bugs and errors in both my classifier and my random generator.