What techniques, tools, or methodologies did you use to assess the overall quality of your code s... (r0_CodeQual)

Local tests, as well as smoke tests. (r1_CodeQual)

I used the coverage tool provided by yarn to assess how many lines and methods my code covered. (r2_CodeQual)

testing coverage and linting (r3_CodeQual)

coverage, my partner and I had a large number of tests which we believed covered about everything. So we were fairly reliant on our C0 test cases passing. (r4_CodeQual)

I used auto test and aimed to get proficient for each checkpoint. I felt like that checkpoint was complete once reaching proficient although if a smoke test still hadn't passed I wanted to figure it out. (r5_CodeQual)

We did lots of testing (ensuring > 80% code coverage) within each checkpoint, and looked at the autograder's feedback for both #check and the checkpoint feedback to see how we were doing. (r6_CodeQual)

Autotest giving me a 100% or Proficient grade and my personal judgement. (r7_CodeQual)

I used the local mocha testing to see if the local tests pass. If they all pass then I move on to the yarn test that we use for this class. If those all pass then I submit it to autobot. I knew a checkpoint was done when I got proficient on the autobot, otherwise I keep working at it. (r8_CodeQual)

linter (r9_CodeQual)

I used static analysis tools, unit testing, and code reviews to make sure my code was up to snuff and to know when I was done with each part. (r10_CodeQual)

Most of it was just an eye check, except for the use of ESLINT. ESLint would give any errors or major red flags when coding, but beyond that for each checkpoint I would look at my code and determine myself how readable it was. (r11_CodeQual)

ESLint and build tools, tests that we've written, autobot, console logging outputs, debugger, ChatGPT (r12_CodeQual)

We determined we were complete with our code when autotest gave us a the highest grade bucket. Checking during development was a mix of created tests and checking our work against the autograder early to see where we need to focus on. (r13_CodeQual)

we tried to have many tests for each checkpoint and felt confident that when all our tests passed and we had achieved the Proficient grading bucket that we were happy with that phase of the project. My partner and I knew we could aways keep working and improving but with limited time and competing priorities it was helpful to receive the smoke test results + Proficient bucket + own passing tests to feel we had done enough for a certain checkpoint. (r14_CodeQual)

local tests and autograder tests. I determined if I was finished with the checkpoint when the autobot reached 100, or I mentally had enough. (r15_CodeQual)

I would mostly add more tests (and revise code to make them pass) and judge my progress based on the autograded feedback. Once I hit proficient then I considered the checkpoint completed. (r16_CodeQual)

Testing (r17_CodeQual)

I used the AutoBot to determine when I was finished the checkpoint. I made sure to document all of my code and make it look clean upon submission. (r18_CodeQual)

Own test suite and auto bot result (r19_CodeQual)

Test suite, code coverage, PRs, complete when reached 100% on auto grade (r20_CodeQual)

Unit testing to access code quality (adherence to spec), and autotest to determine completeness (r21_CodeQual)

When autograder says proficient. (r22_CodeQual)

I used a lot of unit tests and the autograder feedback to determine when I was completed with a checkpoint. (r23_CodeQual)

I wrote an obnoxious amount of tests for every helper and integration/delegation function and I used code coverage to know that I was fully covering things with the work that I did. I also made sure not to only rely on the C0 query tests to provide coverage since they were thorough but often too complex to identify the root of issues when something was caught. (r24_CodeQual)

Proficient in autograder. (r25_CodeQual)

IDE, test suite (r26_CodeQual)

yarn test, yarn cover, ran tests in mocha and chai to ensure all tests passed. (r27_CodeQual)

Type-driven development, unit tests for the rest. (r28_CodeQual)

I run tests to assess my code firstly and after everything is done I used auto test. I determine if I passed the checkpoint by calling auto test. (r29_CodeQual)

I used the tests I wrote in C0 to asses how well the code is performing, but primarily relied on the autograder. I stopped once I hit proficient and my local tests were passing, even though the local tests usually were all passing before hitting proficient. (r30_CodeQual)

The best way to check for us was with our tests. We would write our tests and run #check to make sure our tests are correct and then start implementing till the tests passed. We determined if we completed each checkpoint by running autobot until we got Profficient. (r31_CodeQual)

When the AutoTest said I had full marks, I completed the checkpoint. (r32_CodeQual)

Unit testing mainly. Debugging with debugger, and running coverage tests to see line coverage. (r33_CodeQual)

By running the autobot. Obviously, we had local tests for everything but you can never really be sure and some cases are hard to think about. Autobot gives you a sanity check about if you've been relatively thorough enough ig. (r34_CodeQual)

I wrote unit tests and used both those as well as the integrations tests to make sure my code worked. My partner and I also went over the code together to make sure it was up to standards and there were no style/design issues that one of us didnt like. (r35_CodeQual)

Using the user stories and agile method i was able to assess the quality of the code. Using the methods we learned in class to maintain clean, changeable code with little to no code smells assisted in easy refactorings and extensions. Each checkpoint we would list out the various steps according to the spec. (r36_CodeQual)

i used built-in debugger, test cases, and auto-bot to assess the overall quality of my code solution. (r37_CodeQual)

Checking the autograder score. (r38_CodeQual)

I used library documentation, ChatGPT, an online drawing tool, and Google for development. If the 310bot graded my work as "Proficient" and my solutions matched the checkpoint requirements, I considered it completed. (r39_CodeQual)

I mostly based my self assessment on the feedback of the autograder. (r40_CodeQual)

During development, aside from the autobot submissions/calls, we used mocha and chai for unit testing. Also, access to the playgrounds for querying and the html converter were especially useful. \We relied heavily on the autobot for completion. (We did complete our own tests before submitting to the autobot though) (r41_CodeQual)

Normally before each checkpoint I would create a strong test suite that would evaluate if my implementation is behaving correctly. This test suite would be based on the description provided by us and would try to really target most difficult cases. After that I would submit against the autobot and if autobot said it was mostly okay(buquet of 80%) I would continue working but just in smaller amounts (r42_CodeQual)

We just finished what code we thought would work and then got it autograded. In some cases, we did manual testing to ensure our code behaved like how we thought it would. As for quality, we just made sure things worked correctly and that's it. (r43_CodeQual)

Using the autograder mostly (r44_CodeQual)

calling autobot to see which bucket we are in. Testing locally a lot. I determined that I was done when autobot gave me proficicient. (r45_CodeQual)

when i passed all the smoke tests from the bot, i considered the checkpoint to be done (r46_CodeQual)

Using things like linters, checking the line count, etc (r47_CodeQual)

ChatGPT for when we didn't know how to write specific parts of code. We were done with each checkpoint when the autobot gave us a score of "Proficient" (aka full marks). (r48_CodeQual)

If we got proficient we stopped working (r49_CodeQual)

I used local tests and console.log statements to assess the quality and determine the progress / whether I am done or want a grade for the code so far. (r50_CodeQual)

Local test and remote test(e.g. autobot). (r51_CodeQual)

I mostly just tried to adhere to SOLID principles and to write documentation for my code modules. I decided I was completed with each checkpoint once the buckets were all filled and my code was clean and documented. (r52_CodeQual)

'- We made intermittent commits and ran #check / #c* as often as possible.\- When we reached 100%, we called it for that checkpoint. (r53_CodeQual)

Tests we wrote, smoke tests from autobot, validating our tests against #check, feedback from autobot. We determined a checkpoint was completed when Proficient was reached and/or all smoke tests passed, if possible. (r54_CodeQual)

My team mainly relied on local tests to validate the correctness of our implementations. Only when the local tests passed, did we submit it to github. (r55_CodeQual)

Mostly local testing, but the autobot and its automatic container tests/warning were helpful (r56_CodeQual)

I wrote local tests and ran them both locally, and ran code against the autograder. I determined each checkpoint to be complete when all the local tests pass for a variety of inputs, and the autograder gave proficient as a result. (r57_CodeQual)

Testing, solution should pass my tests and autotest. Once I got proficient on my portion, I was done (r66_CodeQual)

For C0, I relied mainly on the autograder since it was about test coverage, and determined I was done when I had killed all the mutants necessary to get 100%. For C1 and C2, I relied on a combination of the autograder and my test suite from C0, and determined I was done when I was passing all my tests locally and had 100% on the autograder. For C3, I relied on the test suite for the endpoints and determined I was done with that portion when I had 100% on the autograder and was passing all tests locally, and for the frontend portion I just manually tested in browser and determined I was done when there were no errors and everything behaved as expected. (r58_CodeQual)

Wrote local tests before submitting to autograder. (r59_CodeQual)

'-Autobot grading\-Writing tests (r61_CodeQual)

the smoke test results returned from the autotest was used to assess the quality of my code - specifically the accuracy of the functionality. We determined we were done with the checkpoint after getting the highest bucket grading (r71_CodeQual)

I relied heavily on test driven development to ensure that my code was semantically correct. Specifically, I tried to write very thorough integration and unit tests such that if I was all green I could be pretty confident my code was correct. Of course, code that passes tests can still be smelly. So quite often after producing something that worked and often after taking a break to return with fresh eyes, I would refactor my code in various ways to make it cleaner and easier to understand.\\I determined when I was completed each checkpoint by verifying that the autobot placed the code into the proficient bucket and that I had received a confirmation email after submitting any non-code tasks. (r63_CodeQual)

I utilized Github Copilot to assist with unfamiliar TypeScript syntax and built-in functions, and also consulted GPT for assistance when encountering bugs. Typically, I would run the existing test suite before requesting the 310 bot to evaluate my work. (r64_CodeQual)

Testing, autograder feedback, info from IDE. (r65_CodeQual)

I wouldn't say there were tools to assess the code quality itself. This being said, writing a whole bunch of valuable tests always seemed to help. The feedback from the autobot was also often rather informative. (r60_CodeQual)

the main tool for c1 and c2 was unit tests, but for c3 I mainly relied on the autotest feedback for the results. Each point of significant progress was decided by the autotest grade tho (r67_CodeQual)

AutoTest and local tests (r68_CodeQual)

We tried to follow the guidelines for techniques in lectures. We usually decided we were finished with a checkpoint after we got a full or reasonable grade. (r69_CodeQual)

We mainly relied on code review - ie. asking our partner to look over our code and see if it was understandable and well-specified. If it wasn't, then we tried to refactor to make things clearer. We relied on the autograder to determine when we were done with a checkpoint, as well as when our program passed the tests we specified (according to the reference UI). (r70_CodeQual)

I employed a mix of unit testing, code reviews, and linting tools to maintain code quality. Unit testing ensured individual code components worked as intended. Code reviews with peers helped catch potential issues and improve readability. Linting enforced consistent coding style throughout the project. Checkpoints were considered complete when all tests passed, reviews were addressed, and linting errors were resolved. (r62_CodeQual)

Autobot smoketests (r72_CodeQual)

We used autobot grading to verify the current status of the project. Based on the problem area provided by the feedback, we would improve our test suite with the hopes of catching the bug. We decided that we were complete with each checkpoint when we either could not figure out a method to further improve our test suite to catch any bugs or if we reached "Proficient" (r73_CodeQual)

During development, we wrote unit/integration tests (both black box and glass box) to test our code, as well as coverage tools to assess our tests. We used the Autobot feedback to determine when we were complete with each checkpoint (receiving a proficient grade) (r74_CodeQual)

Smoke tests passing was our biggest indicator of determining completion of checkpoints. We also used unit tests to ensure code quality using the check command. (r75_CodeQual)

during development, as long as the other partner understood what the code does, it was sufficient. towards the end, we tried to make sure it was understandable overall. the checkpoints were done when we either got 100% on autobot or when we believe we have satisfied the specification to our best ability. (r76_CodeQual)

I summarized the key points in the specs before starting and after completion, I checked if my code met all the key points I noted down. I also cross-checked with my partner a lot. (r77_CodeQual)

Autograding and local tests. (r78_CodeQual)

We used our own test suite (glass-box and black-box tests), feedback from the TA, and the auto-test feedback (smoke test feedback, #check feedback, and the grade feedback) to asses the overall quality of our code. (r79_CodeQual)

Testing, pair-programming, assistance from TAs, slides, and google searches for further questions were used to determine the overall quality of my code solution during development. When the defined functionalities discussed before the code build were completed, I deemed the particular solution complete per checkpoint (r80_CodeQual)

I wrote integration tests for my code and used AutoTest once my code passed my local tests. (r81_CodeQual)

we made our own unit tests. Postman for API endpoints, but primarily relied on the 310 bot code smoke test feedback and coverage reports (r83_CodeQual)

I used my own and partner's test suite to get a general idea of whether or not the code was behaving as expected. In times where I was unsure about the desired behavior described in the spec, I wrote additional tests and ran them against the solution using #check to verify how to handle those (especially where the spec instructions were not explicit enough for me to understand what to do). (r82_CodeQual)

When we can no longer make meaningful auto grade mark improvements (r84_CodeQual)

Autotest and unit tests. Determine completion when Autotest says Proficient (r85_CodeQual)

I mainly used unit tests. The Reference UI was very helpful for generating test cases and made writing them significantly easier. Generally, when the test cases passed I knew that the code was likely to fall into at least the developing bucket. (r86_CodeQual)

I used tests. I used the grading bot to see if I was done with a checkpoint. (r87_CodeQual)

I used the 310 bot to autograde the commits to main, and if the auto bot says we are at "Proficient", then I determine that I am completed with the checkpoint. I also use code analysis tools and compiler checkers to make sure that all my code can compile. I also do unit testing to make sure that the functionality of the code is in tact. (r88_CodeQual)

Discussing with my partner and ta. Auto bot grades helps us determine the completeness. (r89_CodeQual)

used autograder, yarn tests and yarn cover (r90_CodeQual)

A little local testing, a lot of "feel", and the rest on AutoTest. (r91_CodeQual)

I used the auto response feedback, and fixed what did not work. (r92_CodeQual)

I used the bot checker. (r93_CodeQual)

I used my tests as well as running through them with the debugger to do initial validation. I determined when I was finished by using AutoTest. (r94_CodeQual)

Running the tests to see if the behaviour of what I was developing was working. Also, to debug I only used console.log. (r95_CodeQual)

I used my self-made tests mainly to assess the quality of my code at each checkpoint. The main way I would determine I had completed a checkpoint would be making a pull request of my changes and using the autograder (r96_CodeQual)

Mocha/Chai tests, matching it with the spec, autograder. (r97_CodeQual)

I pretty much just used the autograder tbh. If the autograder smoke tests pass, then I consider myself done. (r98_CodeQual)

The autotest grader and the tests we created from C0 mainly, along with new tests we add as needed. (r99_CodeQual)

I made sure I had completed all points on the specification page. Whenever I got Proficient, I determine that the checkpoint is done (r100_CodeQual)

1) Relied on self-made tests\2) Relied on feedback from autograder\3) Ran test coverage and checked how long tests run locally after certain code changes (r101_CodeQual)

My partner and I made sure to collaborate and communicate constantly to ensure we were on the same page. We were able to determine we completed each checkpoint when we got full marks when calling the 310 bot to grade. (r102_CodeQual)

I use peer review to assess the overall quality of my code. I would write my own test set, and if all my tests are passed I would consider I have completed my part of implementation. (r103_CodeQual)

I used unit testing to figure out if individual components were working. After I was convinced, I would send it to the grader for more feedback and to see if I was done (proficient grade). (r104_CodeQual)

Submitted for autotest grading once all local tests passed and all development aspects were complete (e.g. both dataset and query functions). If we got Proficient and all our tests passed, we determined we were done with the checkpoint, even if we didn't pass all the smoke tests. (r105_CodeQual)

I use local tests to do the majority of testing (and local user simulation trials for C3 front-end). I submit to autotest when I feel ready, and if autotest gives a satisfactory response (ex. "proficient") then I determine that I am done for the checkpoint. (r106_CodeQual)

Testing was the major technique I used to determine when you were completed with each checkpoint. The idea was to come up with as many test cases as possible and pass them all before doing an autobot command to ensure that I met the specification. Moreover, using linter was helpful in understanding that I was following the code standards of the course. (r107_CodeQual)

Local tests and auto tests (r108_CodeQual)

Usually, a comprehensive testing suite was written after a MVP was produced to ensure that all edge cases have been considered. Once all testing suite and regression tests were passing, we pushed our implementation to the autotester and worked with the areas in which we were lacking to improve our implementation. (r109_CodeQual)

We used some unit and end to end tests. Also used an autograder. We determined that we were completed when we had reached 100% on the autograder. (r110_CodeQual)

Refactoring code and making sure there are no code smells as I am coding (r111_CodeQual)

I used ESLint and `yarn test` to make sure that the code passed all test with good quality. I used result from the bot at Github to determine if I completed the checkpoint. (r112_CodeQual)

writing tests, autograder, peer review (r113_CodeQual)

Make sure I could read and understand what's going on, and that there are easy ways to extend it\\I was done when I got proficient (r114_CodeQual)

Frequently tested against my own local test suite, used autograde feedback check and it helped pinpoint the area I needed to focus on like if it was add, remove, list. (r115_CodeQual)

by testing it. Normally I will try to manually test it first, like using postman, sending a request, checking the data directory, run just 1 test, etc.. Then I run the test suites and if it passes then I will make a PR. I determine its completed if it passes the cpsc310 checkbot.. (r116_CodeQual)

Checked our grade with the autograder to see if we were completed with each checkpoint. Ran local tests and also checked with project partner to see if code was readable/understandable and well written. (r117_CodeQual)

'- determined when I was complete when received Proficient at each checkpoint\- used local tests, autograder #check, and linting to asses quality of code (r118_CodeQual)

Local testing then smoke test. (r119_CodeQual)

We used our local tests combined with #check as well as the smoke tests (r120_CodeQual)

primarily testing, code coverage checks, code reviews with partner (r121_CodeQual)

Smoke tests, and check. If I got 100% on a smoke test I would consider it done (r122_CodeQual)

I used the test suite that I developed to assess overall quality during development and used AutoTest feedback to see if I was done with each checkpoint (r123_CodeQual)

Code smells and code coverage, completion determined by autotest (r124_CodeQual)

My own tests and autotest (r125_CodeQual)

The grade and the feedback from the auto grader and the specification. (r126_CodeQual)

The Autotest was the primary determinant of our code quality and checkpoint. (r127_CodeQual)

I relied on my test suite and auto test suite to access the overall quality of code. (r128_CodeQual)

the tests against test cases (r129_CodeQual)

Unit tests, autograder tests. We decided we were done when all of our local unit tests were passing, we achieved an acceptable score on the autograder, and any remaining bugs were fixed (or at least identified). (r130_CodeQual)

code quality was determined using visual inspection and tests\we determined when we were completed by checking the autograder after completing the spec (r131_CodeQual)

local testing and asking LLMs and friends for edge cases. (r132_CodeQual)

Used the tests created to assess the overall quality of the code. When determining if I completed each checkpoint, I used the autograder provided. Once I received proficient on each checkpoint, I determined I was finished. (r133_CodeQual)

We write black box tests before the implementation, so if they all pass, then the majority of the code works. And we also have the auto-bot grader. (r134_CodeQual)

The amount of refactoring we needed to do when adding a new feature was a good estimate of how well designed our code is. The code also needs to run and be correct, which is done through manual testing and the autobot. We used the autobot smoke test coverage to focus. It helps a lot when the bot specifies which thing to focus on, as there's not a lot of feedback. For instance, "Focus on invalid queries" is much better than telling students they did not pass the query smoke tests. The test passes when it passes all local tests for the checkpoint, though we will continue to add test cases whenever they are discovered. (r135_CodeQual)

Unit tests for initial code quality check. Once code coverage was sufficient for auto grader feedback, we would use that. We finished working on the checkpoint once autograder gave a proficient score (r136_CodeQual)

autograder, as long as i get to proficient, i think im good (r137_CodeQual)

We used our test suites for each respective checkpoint to assess the functionality of our code, as well as es-lint. We also used the autotesters #check and #c0, #c1, #c2, #c3 for feedback from the course team and to determine when each checkpoint was complete. (r138_CodeQual)

AutoTest (r139_CodeQual)

a (r140_CodeQual)

I write tests to check the functionality of my code. I determine the time i finish checkpoint by dividing the work and estimating the time for each task. (r141_CodeQual)

Visual studio code, debugger and course dictated libraries/compilers etc. Checkpoints were completed when all the specification was "fulfilled" and autograder returned an appropriate mark (r142_CodeQual)

I normally develop a local test suite for testing my implementation. However, in between development where my code is not entirely complete yet, I often use console logs to ensure that I'm getting the correct output in each place and that my code is going to the correct place. (r143_CodeQual)

I went through the specs and make sure I implemented every required behaviour and mostly used yarn cover to check test coverage. When I've covered 90%+ I called the auto test. (r144_CodeQual)

we peer reviewed our code to make sure the quality was up to our standards. The auto test feedback was a great way to figure out if we were on track or not. (r145_CodeQual)

mostly debugger, some tests. I use autotest to dertemine when I completed with each checkpoint. (r146_CodeQual)

I used the grading bot to determine if I was completed or not, also with writing my own tests (r147_CodeQual)

I would say that I didn't really apply any particular methodology when developing and writing code. It's mostly because: \1. we're not given enough time to absorb a lot of the practices that are introduced in lecture/slides, nor were they emphasized much in any of the checkpoints\2. presumably, most students just want to do the minimal amount of work to complete the project, or at least achieve a satistfactory grade\\That being said, I just used the 310 autograder bot (obtaining proficient) to determine whenever I was completed, with the exception of C3 where my partner and I went through the checklist of points to fulfill for the frontend. (r148_CodeQual)

Used coverage tool given to us, and the feedback that was given through the autograder. We determined completion once we received proficient on the auto grader feedback. (r149_CodeQual)

For c1 and c2 we wrote some test but we used the Autotest result to determine when we were completed with the checkpoints. The feedbacks were helpful, but it would have been great if we knew what error we exactly got! (r150_CodeQual)

I mainly looked at how my code adhered to the design principles taught in class during development, like how coherent each class was. I mainly used the feedback from autotest to see when I was finished with a checkpoint. (r151_CodeQual)

we mark off a checklist of items specified as deliverable by the checkpoint specifications and only consider autobot proficient as complete (r152_CodeQual)

Local tests, manual checks (ie with the query UI given) and smoke tests (r153_CodeQual)

used local testing, then determined if I was completed based on the bucket grading. tried to achieve proficient in the coding artifact for every checkpoint (r154_CodeQual)

Internal tests, chat bot health check first to make sure I pass lint and development builds and then chatbot check for a checkpoint to see what tests I pass or now. I determined when I am done, when I passed to the proficient level. (r155_CodeQual)

The tests we had created were used to assess the quality of solution, then we would test against the auto bot. As soon as we hit proficient on the auto bot is when a checkpoint was complete to us. (r156_CodeQual)

I mainly used intuition to assess the overall quality of my code, if something felt not well put together then I would refactor and change it until I liked the result. I determined when I was done when the autograder told me i was at 100% (r157_CodeQual)

We used the auto-grader's feedback as well as our own test suite to determine how far we're at with our progress. Furthermore, with C3 having a UI component, we also were able to test our project visually. (r158_CodeQual)

Locally written tests and auto test, with check to check that our tests matched spec. (r159_CodeQual)

I heavily relied on my test suite. I wrote a test suite for each class I made to make sure all my functions and helpers were working as expected. (r160_CodeQual)

I generally just use the checker, to determine quality of my code, based on the feedback I would make changes. (r161_CodeQual)

During the project, I used the auto test tools to assess the quality of my code. I determined when I was done with each checkpoint when I go through the specification and check that I have everything that is noted in the specification (r162_CodeQual)

1. Test suites written in C0 and its follow-up developments as more codes are added. 2. By running a few simple cases and manually evaluate the results. 3. Use the smoke test provided by the autograder. \2. By checking the results from the smoke tests. (r163_CodeQual)

bot feedback (r164_CodeQual)

We wrote local tests to check our program first, if all tests are passed, we will call AutoTest to give us some feedback. Then we will improve our program based on the feedback. (r165_CodeQual)

Unit test (r166_CodeQual)

I used an online typescript compiler to make sure everything worked as I wanted it to. I compared outputs on the typescript compiler as well. I determined I was completed through the autobot giving a proficient score. For C3, I determined I was done by checking if the front end website checked off every thing as listed by the specifications for C3. (r167_CodeQual)

ESLint is the primary tool assigned to us for linting our code, and it was helpful for most cases. Other than this, we ensure code quality via code review, and I personally tried to compare our implementation to some of the patterns discussed in lectures.\For c0 to c2, we determined completion by achieving a high enough overall score on the auto grader, although we do take notes of potential improvements and refine them in the next stage. c3 required more investigation of how our application behaved and what the specification required, and we ensured each entry of the spec is fulfilled before claiming completion. (r168_CodeQual)

I used a combination of code reviews, automated testing (using tools like Jest for unit testing), and linting (using ESLint) to assess code quality. I also followed TypeScript's strict type checking to catch potential errors early.Each checkpoint had predefined acceptance criteria derived from user stories. I considered a checkpoint complete when all acceptance criteria were met, and the code passed all tests and linting checks. (r169_CodeQual)

unit test (r170_CodeQual)

Autotest feedback; Local tests (r171_CodeQual)

Unit tests and the GitHub bot (r172_CodeQual)

This project has not helped me at all with my development ability, nor has it made me better at it. So, after following most of the standard conventions, a good branching strategy, and DRY, I didn't care about the quality of my code. (r173_CodeQual)

breakpoints, debugging statements, and specific error messages. I used the auto grader grade as the sign that the checkpoint was completed. (r174_CodeQual)

I heavily relied on the autobot to determine when I felt like my checkpoint was completed - being able to see my progress through discrete measures that the autobot provided gave me a good goal to work towards! (r175_CodeQual)

I used yarn test and autograder bot feedback. When the smoke test results returned full marks for all rubric items, we deemed the checkpoint complete. (r176_CodeQual)

I write tests and use the autograder for feedback (r177_CodeQual)

We used the autotest bot and local tests and in both major checkpoints (c1,c2) we were quite sure that we were finished prior to our last submissions. (r178_CodeQual)

I always write some tests before implementing the features, and start implementing until the test suits pass, and then I will add more tests based on how I implemented, and make sure all the tests are still passing. After I have finished all the tasks in the checkpoint, and I still have some time, I will do some refactoring to the existing implementation, and after making sure they are passing the test suites I already build, I will ask my partner to provide some tests (given the division of work allows us to not knowing the implementation details of each others work before the final completion). (r179_CodeQual)

I would use @310-bot #check and #c0, #c1, or #c2 to check if I have completed all the grading buckets. Otherwise, I would also try running the tests locally first, use yarn build, and use yarn start. (r180_CodeQual)

'- the results of the linter\- by constantly referring to the spec and ensuring I could cover all corner cases\- by writing and passing tests for said corner cases above (r181_CodeQual)

Unit testing and running the eslint. I decided I was completed with a checkpoint when I had finished implementing the solutions, I passed all tests locally and passed as many tests as possible on the autobot before the deadline (r182_CodeQual)

mainly unit tests and smoke tests (r183_CodeQual)

local tests. yarn cover (r184_CodeQual)

I ran my tests, if those worked then I checked it with the autograder and then ran the smoke tests, I determined that I was done with the checkpoint when I reached the maximum grading tier (r185_CodeQual)

testcases, autbot testcases (r186_CodeQual)

We use our own judgement, our own test suite, the feedback from the autograder, and the bucket grade that was given to us via our code repo. Generally we were done after reaching proficient or exemplary depending on the checkpoint phase, however sometimes we added some extra just for some self assurance. (r187_CodeQual)

Try different integration tests; let my partner read and see if she can understand without explanation.\If everything makes sense to both of us, we call the autobot to get a grade. If that's proficient, we are done :) (r188_CodeQual)

Mainly through the use of tests and also manually checking certain features. (r189_CodeQual)

Used local tests we created along with the smoke tests from auto-grader. (r190_CodeQual)

I wrote my own unit tests in my code repository and used the AutoBot smoke tests to confirm that my code was correct. (r191_CodeQual)

I assessed that I was doing well via smoke tests, my own local tests, and by doing a manual inspection of functionality. I also made use of examples online as well as conferring with AI tools if I found my work looked really bad. The finality of whether or not I was done honestly boiled down to the results I got from the autobot. (r192_CodeQual)

Well I don't think I understand your question thoroughly, but for c2 (for the HTML parser), I used the help of the TypeScript sandbox as it was quite quick and I didn't have to run the 'yarn build' and 'yarn run' and 'yarn test' commands on my laptop which took time (maybe because I ran them on WSL on my i5 10th Gen 1.0 GHz processor). I basically used that to verify my logic. I only actually used AutoTest at the very end because I was late on finishing my portion of the project (the HTML parser) so I just relied on that sandbox for verifying certain small aspects of my recursive functions of my code. Also, later on, I just used the debugger of IntelliJ IDEA Ultimate edition to verify my solution. I specifically used that because the code was timing out (apparently taking more than 10000 ms) and so I delayed wasting my AutoTest feedback calls. Instead, the debugger (when I stepped through the steps my code performed) did not time out and that helped verify the 'logic' of my code. (r193_CodeQual)

Writing lots and lots of tests for each method. \Google search.\chatGPT.\I determined that I had completed each check point by making sure all my tests were passing locally and then testing them against the auto grader (r194_CodeQual)

Quality is assessed through making sure the code follows the spec, including handling of edge cases. Also reviewing each other's code thoroughly to catch errors/missed cases early on\Determined completion for each checkpoint after all smoke tests pass (r195_CodeQual)

using the auto bot to see if reached proficient (r196_CodeQual)

I use 310 Autobot to assess the overall quality. (r197_CodeQual)

just used yarn lint, determined when completed checkpoint when we reached the 100% bucket (r198_CodeQual)

When the autograder was 100% and non-coding tasks were complete is when I assessed I was done with each checkpoint. (r199_CodeQual)

We mainly relied on our own suite of unit + integration + end-to-end tests, as well as the linting & autotests provided by the course. (r200_CodeQual)

The overall runtime of my code and ESLint. (r201_CodeQual)

We looked at the autotest feedback for each #c1, #c2, #c3. If the feedback states that we were proficient, we considered it completed. (r202_CodeQual)

Running tests locally was how I mostly ensured that my code was working as expected. The autobot was what I primarily used to tell if we had completed the checkpoint, or how far off we were from the checkpoint through the smoke tests. The check command was also very helpful in determining where errors were occurring in our code. (r203_CodeQual)

Writing tests and discussing the code with my partner. Also payed attention to runtime. (r204_CodeQual)

For C1 and C2, I created unit tests to ensure that the methods I created were working as intended. I also created tests and example queries to test performQuery as a whole, using many different types of queries to ensure that all cases were covered. (r205_CodeQual)

By running the checks on Autotest and looking at the feedback there. (r206_CodeQual)

Used chatbots to diagnose issues with code and to come up with ways to initiate an implementation for the code. I determined it was completed by using my local tests. (r207_CodeQual)

I used my local tests alongside the auto grader to assess code quality and when I was 'done' with each checkpoint (r208_CodeQual)

Mainly black-box test-driven development and prs. Completed would be associated with mark. (r209_CodeQual)

If it passed the unit tests/smoke tests, we knew we were pretty much done. Re-read the spec many times. (r210_CodeQual)

For each checkpoint this was slightly different. Overall I looked at test cases passing, coverage of my tests, and then the basic smells we were taught in class. Then I would just send in my code to the grading and use that for feedback. (r211_CodeQual)

I used the @310 bot and #c? to check my progress during checkpoints (r212_CodeQual)

intellij. From developing experience. (r213_CodeQual)

I determined I was finished when the bucket grading said I got 100% (r214_CodeQual)

Extensive testing, and checking with AutoTest (r215_CodeQual)

most IDE, I used IDE's debug tool a lot (r216_CodeQual)

Made sure, I was following material we covered in class and feedback from TA's (r217_CodeQual)

Locally test first, call 310bot. Based on the feedback, add more local tests. And checking the posts on Piazza for the similar issues. (r226_CodeQual)

I would have my own version of a "checkpoint" by dividing my work into small tasks and writing tests to ensure that my implementation is correct. After doing so, I also run the check with the 310bot to make sure everything is on track. When I finish a big milestone or checkpoint, I call the 310bot to test the code. (r218_CodeQual)

Tests (r219_CodeQual)

local tests (r220_CodeQual)

I used the autograder, my groupmates, and feedback from my TA the most to assess the quality of my code. \For a lot of the project, I was quite new to using the tools given to us, so I had to ask my team if what I was doing was correct. \\I tried to look over things myself and apply some of the concepts we learned in class (i.e. design patterns), but I found it to be hard and costly on time to make these changes by the time we got to some of them. \\I determined when I was done by checking the autograder, asking my TA, and doing my own review on if my code was understandable by me and my teammate. (r221_CodeQual)

copilot (r222_CodeQual)

Majorly depended on the smoke tests and the passing (r223_CodeQual)

I tried to use the principles we learned in class, and writing notes of what each function and class should be doing to assess the quality of our code solution. (r224_CodeQual)

Clearly, Autotest is the main tool we used for testing what we have for each submission. We usually aimed to make each part gain as many points as possible at the first step and fix the minor bugs in the end. For each checkpoint, if we achieved the most points and the rest of the bugs were complicated, we usually say we were done . We used a lot of #check on the later half of the project but the responses we got were not really clear or easy to understand sometimes, which made us not sure the problem was about the local test or code itself. (r225_CodeQual)

The overall quality increased a lot with code smell. Just removing duplication and using SRP helped deconvolute the code and made it a lot cleaner. It is complete when each checkpoint in the specification is achieved. (r227_CodeQual)

A combination of my local tests and the autograder. When the autograder showed 100% I would determine I was done with the checkpoint. (r228_CodeQual)

I just used autograder to determine if I was complete. Also sometimes my own unit tests. (r229_CodeQual)

Local tests and autograder feedback (r230_CodeQual)

The bot tests determined when we were done. In order to keep overall quality I tried to look at my code after a break and see if there's anything I could make more reusable/readable and reduce code duplication from shotgun surgery (r231_CodeQual)

Writing tests, manual tests. (r232_CodeQual)

Mainly Unit Tests and what was explained in class (r233_CodeQual)

Multiple Test, TA Help (r234_CodeQual)

I mostly relied on writing test suites and called autograder. I tend to not develop a large number of tests beforehand and add tests in when autograder gives feedbacks on some particular part of my implementation. (r235_CodeQual)

Used Mocha test suites to verify that the code was functioning correctly. We wrote the test suite first, so once all the tests passed, it was deemed complete. Also compared code results to the reference UI results for manual testing. Used eslint to check for style issues, and completion was when eslint reported no errors. (r238_CodeQual)

For some of the initial checkpoints, our test for quality of code was just how seamlessly the code worked, and whether a single class needed to be changed for additional features or changes. We assessed the quality of our code to be poor in cases where multiple classes had to be changed to implement a small change in the code. As the term progressed, we started focusing more on the design principles taught in class, such as the INVEST principle, and made sure the code adhered to most of them. (r236_CodeQual)

Unit testing mostly, and then running against autotest. (r237_CodeQual)

Run all the yarn commands\Check coverage\I was done when the autograder gave me 100 (r239_CodeQual)

used my personal test suite (r240_CodeQual)

Writing our own tests helped determine the quality. We usually determined we were complete with each checkpoint once we got to a certain bucket in the autograder. (r241_CodeQual)

I mostly relied on units test and manual testing for the desired output and then as a final step using auto test. (r242_CodeQual)

For me, I drew out a blueprint for code structure, the classes and methods and functionalities on my iPad, before implementation. This way it helped me visualize and assess the correlation between classes. As we progressed into other checkpoints, when I encountered difficulty adding new elements to my existing code, that was a sign for perhaps sub-par quality / strucutre of my previous code. (r243_CodeQual)

I used the AutoTest feedback as well as my general programming experience to assess the overall quality of my code solution during development. I determined when I was completed with each checkpoint when AutoTest returned full marks or I could not feasibly devise a better solution within the remaining time. (r244_CodeQual)

For quality of code, I usually thought about how much code smells I had and if I was using best practices. I determined I was done when my tests were all passing. (r245_CodeQual)

autobot, checking with partner (r246_CodeQual)

The main guiding heuristic was "modularity". It's not realistic for novice programmers to try to code while constantly mentally returning to SOLID design principles (let's face it - how often do we think about Liskov substitution principles while coding?) - we're more focused on solving the problem in a decently principled way. Writing "modular" code seems to encapsulate the main gist of SOLID - write code that scales and interprets easily. (r247_CodeQual)

Ensure checkpoint requirements are met to ensure auto test to be fully covered. (r248_CodeQual)

Testing and version control (r249_CodeQual)

Used local testing to assess the overall quality. I determined i completed the check point when both local and autotest passed (r250_CodeQual)

Used test-based development and some LLMs. Completion is determined by when a checkpoint grade could not be advanced any higher. (r251_CodeQual)

Before checking with the bot, I would run my own tests on my implementation, and for c3 I would also test things myself just to be sure, like with Postman. Then when I was pretty sure we had a good solution, I would check with the bot and based on the feedback, go back and fix or update the code. At each new checkpoint, we decided to look through our existing code and see if any refactoring strategies that we saw in lecture could be applied to code smells that we found, and did some refactoring accordingly. To determine if we were completed with the checkpoint, we not only checked with the bot and our tests, but also read over the spec to make sure we didn't miss anything. (r252_CodeQual)

To assess the overall quality of my code, I usually just checked coverage and ensured that the autograder returned the maximum mark. I also made sure of course that the linting was perfect. (r253_CodeQual)

gbd, using the smoke tests to gauge how close we were to checkpoint. (r254_CodeQual)

I used self-checking and the autograder to determine the quality of my code. I stop when the deadline pasted or the autograder gave me the score I was satisfied with. (r255_CodeQual)

We stopped once we got to the proficient bucket and when at least 80% of the local tests were passing. (r256_CodeQual)

During each checkpoitn, we would assess the overall quality using the SOLID framework and the autotest feedback. (r257_CodeQual)

I build my test suite and use the test suite on the server as well (r258_CodeQual)

Primarily relied on test suite developed by myself and my partner. Also checked heavily with autobot for smoke tests results and the bucket grade. We would aim for proficient through each checkpoint so if the autobot said it was proficient we would be satisfied. (r259_CodeQual)

We would work until the end to attempt to get our project as bug free as possible. (r260_CodeQual)

Talking it over and having my partner look over the code. The linter also helped with making sure that the code followed style guidelines. (r261_CodeQual)

We stopped working once 100% was achieved on autotest. (r262_CodeQual)

I wrote a bunch of tests to make sure my code was all working, and based of what was learned in lecture to refactor the code for good quality. (r270_CodeQual)

'- Tests written in InsightFacade.spec.ts (how many of them passed, how long they took to run)\- running tests locally using 'yarn cover'\- calling #health and #c0/c1/c2/c3 on autobot to check how many smoke test clusters pass (r263_CodeQual)

I mainly used the tests written by my teammates to determine whether the solution I wrote was correct before sending it into the autograder. Sometimes, I would also think of tests that my teammates did not include and I would write them myself as well. The best way to figure out which parts of the checkpoint are complete are to run the autograder (#c1, #c2, or #c3), but since these are limited per day, we only did this after most, if not all, local tests passed already. (r264_CodeQual)

I first checked my code against specifications, using tests written based on the specifications. After a given objective was finished, we'd submit and use the autotest feedback to double check that we were on the right track. (r265_CodeQual)

using auto grader class help me determine if I already completed the check point (r266_CodeQual)

I used criteria we learnt in class to assess the overall quality. Completion of each checkpoint is reported by auto-grader plus double checking the requirement of the none-code artefact. (r267_CodeQual)

Used my own tests to initially assess my code. I usually was content with my code after all tests pass and usually autocode matched this. Sometimes I had to fix a bug if autocode wasn't happy. (r268_CodeQual)

1) Yarn cover\2) Manual reading of the spec reqs (r269_CodeQual)

It was when our tests passed as well as the autograder. I proofread my partner's code to make sure it catches each edge acse. (r271_CodeQual)

purely based on autotest feedback (r272_CodeQual)

Utilized our test suite and auto test feedback, as well as test suite coverage to assess the overall quality of our code. (r273_CodeQual)

Debugger. Using #checks and calls to checkpoints. Running tests personally (r274_CodeQual)

first by making a complete and correct tests, I can see how much of the function is actually working (r275_CodeQual)

Aside from the Autobot feedback, I assessed the workflow and quality of my solution by running through the processes at a high level. Then, I would assess that against the tests to check if there are any inconsistencies. However, most of the evaluation of whether each checkpoint was done was by seeing the feedback and grade on Autobot, and confirming with each other that we were satisfied with our work. For two checkpoints where we did not get our desired grade in time, we would work close to the deadline and do our best to check in with each other, stopping when we feel like we have done our best to complete the criteria expected in each checkpoint. (r276_CodeQual)

I wrote a lot of tests for C0, C1 and C2, which I used the #check feature to verify were correct. I then repeatedly ran my local test suite to ensure that my code solution was solid. I decided I was complete with each checkpoint when my team either reached the proficient bucket, or we ran out of time. (r277_CodeQual)

I wrote some local tests to test the basic functionality of the functions and stepped through code with debugger to find specific bugs in the code when the tests fails. I also use the autograder to test if my tests and code are correct, and if there's problems such as missing files or runtime issues. (r278_CodeQual)

IntelliJ and Yarn to check code coverage. (r289_CodeQual)

I used the mocha testing framework that we set up at the beginning of the project for local tests, and the auto-grader feedback to check if the code met the requirements. (r279_CodeQual)

I determined if I completed each checkpoint by pushing my code to GitHub and calling the Autobot to grade my work since it gives marks. (r280_CodeQual)

I used the autobot to determine if my code was completed (r281_CodeQual)

Testing, coverage testing, code reviews (r282_CodeQual)

The main thing that was used to ensure the correctness and completeness for each project checkpoint was the autotest autograder. It also strongly helped in guiding us through each checkpoint, with it giving exact details of which things we got right and which are on the wrong track. (r283_CodeQual)

I used my mocha and chai tests as well as check with the specifications listed and the autobot for further guidance. (r284_CodeQual)

I just used whatever came pre-packaged with the starter (r285_CodeQual)

When the autograder determined that all the smoke tests passed (r286_CodeQual)

I j just made sure I implemented everything according to the spec, I had a private test suite, although it was far from comprehensive, it gave me confidence in my answer. I used autograder feedback to guide additional tests and find bugs. (r287_CodeQual)

Debugger, Autotests. (r288_CodeQual)

We didn't do this very well which is probably one reason the refactoring in c2 was so significant to our project. (r290_CodeQual)

In developing my code solutions, I employed several techniques and tools to ensure quality and determine when a development checkpoint was complete. Here's a breakdown of my approach:\ \Unit Testing: I frequently used unit tests to verify each function or component independently. This helped catch any errors early in the development cycle. I used frameworks like JUnit for Java or pytest for Python, which facilitated test-driven development (TDD).\Code Reviews: Participating in peer reviews or having my code reviewed by more experienced developers was invaluable. This not only helped identify potential bugs and better practices but also ensured adherence to coding standards. (r291_CodeQual)

local test suites, if they all passed, then use the autograder. (r292_CodeQual)

123 (r293_CodeQual)

I use a combination of the test suite as well as chat bots to determine if my solution is good. (r294_CodeQual)