Code Source

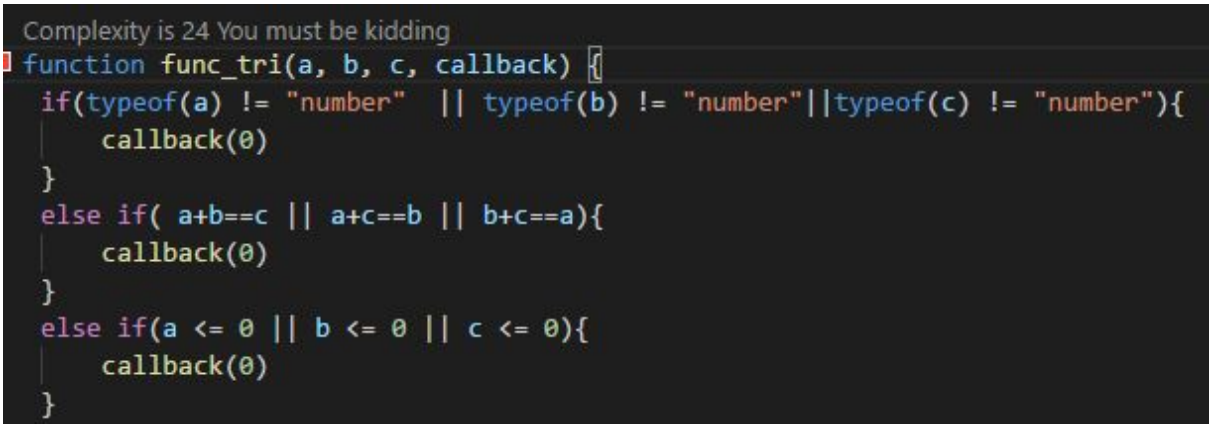
https://github.com/Thug-Lyfe/test_ass_1

2) Static Code Analysis of Triangle Application

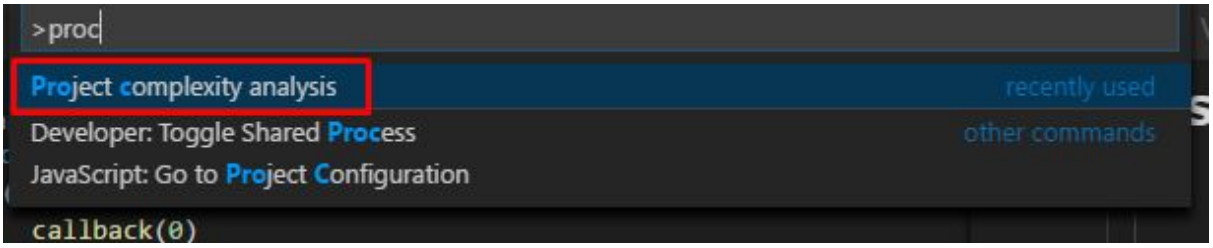
A) I use vs code and js so i went with two different metric extensions, since that was all the extensions i could find, and I wanted to try all my options.
   a) The first one is: ***"CodeMetrics"***, which computes the complexity of functions live, shows it above all functions.
   b) The second one is ***"JS Complexity Analysis"*** which makes a more thorough analysis and can be used to analyse a whole project instead of just individual functions.

B) I have used a couple of coding standards in the triangle project.
   a) The main one is keeping it async, I like async as it gives more versatility to a function.
   b) There is no null return, you get either 0,1,2,3 for "not a triangle", "Equilateral triangle", "Isosceles triangle" and "Scalene triangle" respectively.

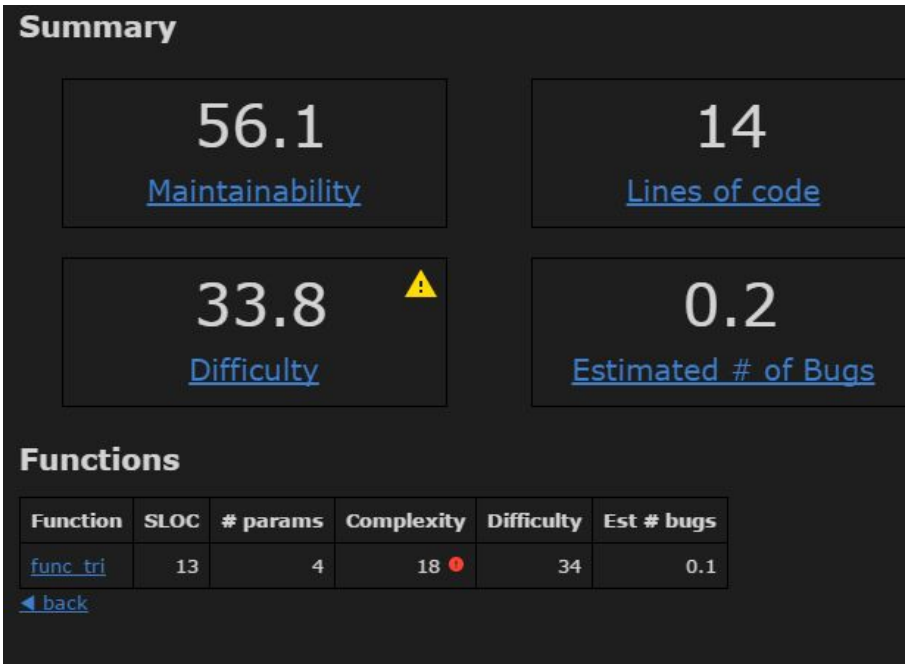
C) As I used 2 different extensions I have 2 outputs, the first printed above my functions is 24 as can be seen here:



The other is run like this:



And is shown like this:



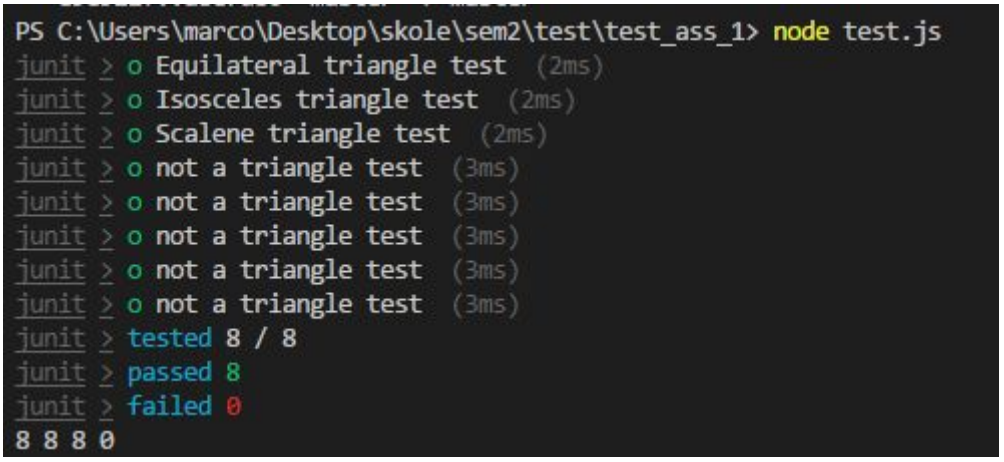
D) I am pretty sure the both measure CC2 as they both take boolean operators into account, but I am not entirely sure.

E) I did in fact not change my code.

```
Complexity is 24 You must be kidding
function func_tri(a, b, c, callback) {
  if(typeof(a) != "number"  || typeof(b) != "number"||typeof(c) != "number"){
      callback(0)
  }
  else if( a+b==c || a+c==b || b+c==a){
      callback(0)
  }
  else if(a <= 0 || b <= 0 || c <= 0){
      callback(0)
  }
```

Now I could have shortened this down to one ***if*** instead of three and it would reduce my CC from 24 to 22. but I noticed that the other extension was unchanged. This in turn made me not change it for other reasons instead, as I like having them separated as it gives more clarity as to what happens in the code instead of just shoving it all into one line.

F) The junit tests can be found in the ***test.js*** file in the repository and is run with ***node test.js*** and should give the following output:



As I had already done some tests on my function while writing the code for it, to make sure it worked as intended. I have merely refractured those tests into junit with the junit npm package not have to refracture anything as I already tested it extensively during the development process, and I have to admit that I just took those test and refractured them into junit.
On a side note remember to npm install before running them to download the dependencies.

3) Peer Review Checklist

Read it, set common goals, have a metric to check against, plan ahead, review segments at a time, take breaks, review prior defects, have fun, don't be quiet, everyday all week, use a tool, have a list and go through it.
That is how i see the smartbear checklist, and my group and I have for the most part used part of it in our projects even though I had not heard of the smartbear list prior to this day. for example: reviewing small parts, having fun and using a list and a set of goals.

A list is always nice, a todo list can do wonders for a project, as it both keeps track of and reminds the developers of what is missing from a viable product, we even wrote defects on it, so that when it was fixed it could be crossed out.
When any part of the code was done it would immediately get scrutinized by the other group members as they would be interested in how the problem or feature was fixed.
through both we would always keep it light and fun so that it would not feel like an obligation but more of a social thing were we could better our product.

These were things we already did and it covers about 40-50% of the list, new things that I would definity use in the future is probably a tool to do the defect finding, and daily reviews.

Other things that could be added to the list could be things like:

Error handling
Making sure the error msg contains enough information to identify, validate and solve it. Making sure it is not a breaking error, for example a web application that throws an error which stops the web app from working is a big no-no.

A checklist of documentation.
Are the developers writing valid and thorough documentation for their code, so that others can properly review it, in an acceptable amount of time.

4) Review of failing tests

I found the problem to be:

```
assertEquals(0, underTest.getNrOfPeople());
```

As removing this line makes the code run successfully in an IDE when copied over.
I don't really know why it fails, the best guess my group and I could get to is that the test is run async, which should not be possible as the jUnit4 documentation states that it should run all tests sequentially.

If you still want the test run, I would use the before function instead so that all tests are runned with a new instance of Catalog.

5) Code standards

The coding standards should always depend on who is developing and for what purpose on for whom it is for.
But some of the more common points that is shared across most developers are:
Format: making sure the code is written in the same way, so that it is not confusing for someone reading it through, this includes things like either keeping it async or not, as many problems occur when one developer is writing with async in mind and another who then gets
Errors: when things are not happening sequentially.
Errors: Making sure that there are no empty or breaking errors and if there they should be fixed as a number one priority. One should always write valid error messages so that they can be located and validated if they come to fruition.
Version control: Always use verified dependencies, and do not just use the most current, as there could be vulnerabilities passed in latter iterations and you cannot be expected to follow all of them at all times.

6) Gitte’s presentation

Now granted I have a very bad memory, but the thing I took away the most was how many different ways you could test and what their implications were.
I also resonated with the idea of doing the arithmetic of the cost of bugs, in regards to when it is caught. This resonated with me alot as I have recently heard a lecture/podcast about a book called the “***12 Rules for Life: An Antidote to Chaos***” by ***Jordan Peterson*** where he as a clinical therapist, mentioned having his patients break down their actions during the day into segments and measure how much time is actually used in a month.
Where small things like fighting with your kid for 30 min each time they go to bed becomes 15 hours a month, which is a lot of time.
Now i related that to coding practices during the presentation where a bug might just cause you a couple of minutes a day now, but if you let it persist, then the time you would have spend on it would accumulate and would far overpass how much time it would have taken to fix it.
And if a bug is then bothering more than one developer/end user, it would cause much more harm.
I also quite liked the idea of specialists who are not just focused in one area as that is also what I see myself in just ordinary life. Like the saying “***Jack of all trades, master of none, but better than a master of one***”.