Assignment 4 - Static Analysis

Version: April 4, 2020

Due: Tuesday, Apr 14th, 11:59pm

Objectives

- Understand how a static analysis tool helps discover potential defects and other quality issues.
- Understand merging in git.

Setup

This is again an individual assignment. You are required to have a document in which you answer questions and include screenshots.

We will continue to work on your private repository.

Some Git things first (8 points)

Lets talk about Git a little. We deleted the "GivenBlackbox.java" from our Whitebox branch since we did not want our tests to fail, which was a drastic move. What do you think would happen if we would (do not do it) merge Whitebox into master?

Answer: Your "GivenBlackbox.java" would also be deleted on master. Which is not really what we want, since we want to keep our test files.

So when you work with Git and you change or delete a file in a branch, that might overwrite things in a way which you do not want it to. It depends highly on who their ancestor is and what changes have been made, which are newer and if they conflict. This will be something that you will run into, I advise you to pull the Dev branch every time before you start working and merging that Dev into the branch you are working on to decrease the likelihood of complicated merges. And remember you can always go back in time. So lets look at how you can get your file back (works for other changes as well).

First switch to your Whitebox branch. Create a new branch off of Whitebox and call it StaticAnalysis. Switch to that branch, we do not want to change Whitebox since that is still being graded. So make sure it does not change.

Do "git log" in your command line which will show you a log of all previous commits on this branch and its ancestors. Look at the commit messages, did you do well and can actually understand by reading them what you did? (rhetorical). If you cannot then please change how you write your commit messages in the future. You will probably get a lot of log entries (if you committed often). So lets filter them by deleted files (other filters are also possible).

```
git log --diff-filter=D --summary
```

This should give you the information of any commit where a file was deleted (in our case that should be exactly one – more if you deleted other files – which you might have done to clean your repo).

Now we can checkout the file we want from a specific commit:

```
git checkout COMMITNR~1 src/test/java/GivenBlackbox.java
```

where COMMITNR is the long commit number given in the log entry from where we deleted the file (the ~1 then says use the commit **before** that specific commit – since we want the file before it was deleted – if you want exactly that version then ommit the ~1). From that specific commit it will get the specified file (it also works for directories if needed). It should look something like this:

```
git checkout b77d051a79624b3e14529c0d330f80b5e79a3d94~1 src/test/java/GivenBlackbox.java
```

If you struggle with this go back to the Git videos from the beginning of class, I talk about these things in there as well.

When you check your explorer/Finder/command line the content of your assignment folder you should see the "GivenBlackbox.java" file again.

Now you need to add and commit this file so it is back in your current version in your StaticAnalysis branch.

We still do not want to run our Blackbox test when running gradle build since our build would fail. Gradle allows us to specify which test files to run. Since we want the change we are doing now in all our branches we are going to get a little "complicated". Pay attention to what exactly I want you to do and follow it exactly.

Switch to your **Blackbox** branch. Now add the following code to your build.gradle file:

```
test {
    exclude '**/BlackBoxGiven.class'
}
```

When you run *gradlebuild* now (do *gradleclean* first) it should only run the specified test classes and your build should be successful. If it is not something went wrong, try to figure it out.

Now, lets merge this into our StaticAnalysis as well.

Switch to StaticAnalysis, merge Blackbox into StaticAnalysis (this should only add the Gradle lines you just added). Running Gradle build on StaticAnalysis should now run all your tests successfully and thus make your build pass. If it is not you should make sure you correct your tests or code in your StaticAnalysis branch.

Cleaning up (5 points)

Your repository should already be clean at this point but if it is not you really need to clean it up now. I do not want to see temporary files in your GitHub repo anymore when you submit the assignment. This means that any unnecessary hidden files and build files should NOT be in your repo anymore. If you do not know what this mean please refer back to the Git videos and read about what should be in a gitignore file. Make sure you delete them and they are ignored through a good gitignore in your master, Blackbox and StatciAnalysis branch (DO NOT change Review and WhiteBox since we might still be grading these). You might already have done this so then there is nothing to do for you here and these points are a gift for already being well organized.

Setup for this Assignment (7)

Travis CI (5 points)

Our goal is that our build is always successful and Continuous Integration helps us with this. So lets set up Travis CI for your assignment which will use Gradle to build your assignment code.

Download the given Travis CI file (.travil.yml) and add it to your **master** branch. You find the file on Canvas with this assignment. Add it to your root directory (same place where your build.gradle file is). Add and commit this file.

Now, we also want Travis CI to use a specific Gradle and Java version. Gradle has something called gradle wrapper which can be used to specify the Gradle and Java version. To use it you need to run "gradle wrapper" in your command line (also check here: `https://docs.gradle.org/current/userguide/gradle_wrapper.html`). This will generate the needed files. Add all generated files but the .gradle folder to your repository (depending on your gitignore it might ignore the .jar file make sure it is included though).

Now, merge these changes into your master, Blackbox and StaticAnalysis branch! This should only add the TravisCI changes. Make sure it does and that you do not overwrite something. You can consider doing this as PullRequest on GitHub to have an easy overview of what will be changed.

Got to `https://travis-ci.com` and login with your GitHub username and password. You should be able to see your assignment project here. If not check on the Travis CI help on what to do.

TravisCI should now build your project whenever you commit something to GitHub and thus will tell you if your build is successful or not. TravisCI will also run your Unit Tests if they are included in Gradle (as they are for your project) so you will see if your tests pass. It gives you the means to constantly see if your build is still working.

Your master branch should fail (since the Blackbox tests still fail and were not excluded from running in this branches). StaticAnalysis and Blackbox however should pass (since we excluded the Blackbox test). Check on Travis CI that it does and include a screenshot in your submission document. You can initiate a build on TravisCI directly in case it did not build that branch yet. Review and WhiteBox should not show up on TravisCI since you should not have changed these branches at all!

So for now that was just to show you how you get a file back,how you can make Gradle exclude a test file and setup Continuous Integration.

Branch (2 points)

So looking at our branches we have master -> Blackbox -> Review, Blackbox -> Whitebox and now Whitebox -> StaticAnalysis. So Review and Whitebox/StaticAnalysis are parallel workflows but decedents from BlackBox.

Commit throughout working on the different tasks whenever you think a commit makes sense. Work on your **StaticAnalysis** branch for the following tasks (non of the other branches changes for now).

Task 1: Style checking using Checkstyle (13 points)

1. Open the build.gradle file from the project and uncomment the things related to Checkstyle (not Spotbugs yet). Commit and push this change (should also trigger a

build on TravisCI).

2. You can use Eclipse or Gradle for working with Checkstyle. One bullet points lists the Eclipse parts but you can completly skip that if you like. It is sometimes nice to see the violations in the IDE though but it is optional.

3. Eclipse things (optional):
   - Install the Checkstyle plugin into Eclipse from the Eclipse marketplace (at least version 8.7.0 for older versions the configuration file - see later - might not work):
     - I just went to the Eclipse Marketplace (in Eclipse Help -> Eclipse Marketplace) and searched for Checkstyle and installed Checkstyle Plug-in 8.7.0
     - Sometimes the installation for Checkstyle in Eclipse directly does not seem to work, if that is the case, you can download a zip from'here (

       `https://sourceforge.net/projects/eclipse-cs/files/Eclipse%20Checkstyle%20Plug-in/`

) and then go to Help - Install New Software and use the Archive option to use the zip you downloaded.
     - Also on Windows I needed to install Checkstyle as admin. It did not work if I installed it without opening Eclipse as admin.
   - Make sure you do the following check: Go to Eclipse -> About Eclipse -> Installation Details. Check which Checkstyle version is mentioned there! If it is lower than 8.7.0 deinstall it there and go through the motion from above again. (We had the problem that the Marketplace said 8.7.0 but in Installation Details it used an older version, which lead to Checkstyle not working). If it shows 8.7.0 then you are good to continue.
   - Setup Checkstyle
     a) After restarting Eclipse, right-click on the project and click 'Checkstyle » Activate Checkstyle'.
     b) Go to the Eclipse Preferences (Mac: Eclipse Preferences; Windows: Window Preferences) and find the Checkstyle configuration. Click the 'New' button in this configuration window. In the dialog, enter the name 'SER316 SA'. Then click Import and navigate to config/checkstyle (this config folder is in the given folder) and import the checkstyle.xml file. This is a checckstyle configuration file.
     c) Set this new Check Configuration as default.
   - You should rebuild your Eclipse project. Uncheck Project Build Automatically, then do Project Build Project.
   - Right-click on your 'src' folder, and do Checkstyle Check code with Checkstyle.
   - In the bottom pane bring up the Checkstyle Violations view (you'll find it under Window Show View Other). You should see a big pie chart with your violations.

4. Checkstyle is already included in the given Gradle file you need to uncomment that part. Checkstyle in Gradle uses a config file which is included in your repository

in the /config directory, make sure you have this directory and it is also version controlled.

   a) Depending on your Java and Gradle version you might want to change the toolVerion from 8.8, which is pretty old and basically leads to Gradle chosing the newest version, 8.31 which was just released. This newest version made some changes though and lead to some issue with the given config file. Using 8.3 worked well for everyone I know.
   b) Look at the Gradle file, you should get a rough overview of what this Checkstyle code does
   c) Run 'gradle build'
   d) Check out the checkstyle report (reports folder)

5. Find the Checkstyle report after calling "gradle build" and take a capture of the report.
6. Answer in your document: How many violations did Checkstyle find in main and how many in test (easiest to see through Gradle reports)
7. You will have a bunch of violations. Most of them will be about indentations and tabs. The Checkstyle style file I gave you assumes that there are no tabs and that the indentations are 4 spaces. You can change settings in your IDE to have only spaces instead of tabs. You should do so (google if you do not know how) and then basically go to each java file and let your IDE correct the indentation. This should greatly reduce the violations you get. How many violations do you have now in main and how many in test?
8. Now, fix violations in the project (all the files in the main package) until you are below 25 violations in that package.
9. When you are done take a screenshot of your report.
10. Take a screenshot of your command line window with 'gradle build' and its output, and the generated HTML file from main on it (also of course your asurite/name somewhere).
11. Make sure that the code is still running correctly (as correctly as the first version was running).

Task 2: Static Analysis using Spotbugs (14 points)

Task 2.1: Gradle

- Add SpotBugs to the given Gradle file (just uncomment what is commented for it - there is something at the top and something at the bottom you need both) – depending on your Gradle version it might throw an issue. Ask Google how to fix it if needed.
- run "gradle build"

- Gradle should have created an HTML file for you with a couple of bugs (in reports folder).
- Take a screenshot of your report and command line window (and asurite) and put it in your document.

Task 2.2: Fixing the bugs

PART 1: Part 1 is optional and just needed if you want to use Eclipse.

1. You can find SpotBugs in the Eclipse Marketplace, just search for SpotBugs and install
2. Change settings for SpotBugs:
   a) Go to your projects "Configure Build Path" then Spotbugs
   b) set the "minimum rank to report" to 20
   c) Select ALL "Reported bug categories" checkboxes
3. Run SpotBugs:
   a) You need to make sure your project is build
   b) Right click on your project and look for SpotBugs and start it
   c) It will say that the project is partially compiled but it worked for me
   d) You should be asked to switch to the SpotBugs view, do so and look at the bugs found

PART 2: This part everyone has to do.

1. Fix ALL errors reported by SpotBugs, add the comment 'SER316 TASK 2 SPOTBUGS FIX' in all CAPS in front of your fix – or on the same line if you prefer
   - Hint: Almost all of the bugs that are being reported by SpotBugs have a stack overflow post with exactly the same description which pops up in the bug explorer of SpotBugs. This will help you correct these errors.
   - Hint: If you have the HTML file you can click on the bug and find out some more information about it.
2. Take a screenshot of either your HTML file and command line or of your XML and Eclipse environment showing SpotBugs run and add it to your document (of course showing your asurite somewhere as well).

Task 3: Comparing (3 points)

Now we have done the Static Analysis after our Whitebox Testing but not after our Code Reviews (but in parallel). So lets first check how you did without Static Analysis and if your Code Review already improved the code.

Do the following for your Review and Blackbox branch.

Switch to your Review/Blackbox branch and uncomment all the Checkstyle and Spotbugs things in the gradle file (commit and push this change).

What do you get now for these branches? Write in your document how many Checkstyle and Spotbugs violations/bugs you got and compare to the **initial** report you got from your StaticAnalysis branch (in main and test for Checkstyle and Spotbugs). How does this compare to the violations on the StaticAnalysis branch at the end of correcting the violations and bugs. Make a nice table in your document showing all this data for your different branches (Blackbox, Review, StaticAnalysis before all your fixes and after). You do not have to change the code in the old branches (eg. do not correct the violations/bugs).

Answer: Did it get better or worse? Can you explain why?

Task 4: Putting it together (5 points)

Now let's put things together. This might get a little tricky since you worked on (almost) all the files in the "parallel" branches StaticAnalysis and Review. So you will probably get a good amount of merge conflicts.

Start off with creating a "Dev" branch which is based off of Blackbox (so we pretend we have a rather old version in it). Switch to this Dev branch and now merge Review into it. Resolve all the conflicts manually if you have to.

After resolving all conflicts, check if your code still works as intended.

Now, merge StaticAnalysis into Dev. Again resolve all conflicts.

Now you should have a branch which has all your work in it.

You should make sure that you will have all your tests cases/classes included (including BlackboxGiven.java), that your Gradle file excludes your Blackbox test and has Checkstyle and Spotbugs activated.

Run gradle build on this Dev branch and check the number of violations. How many violations do Checkstyle and Spotbugs report (write the answer in your document).

Answer: Did it get better or worse and can you explain why?

Reduce the Spotbugs violations to 0 and your Checkstyle violations in the main package to below 25 again (if they are over this threshold).

Run all your Unit Tests (excluding Blackbox of course which should still not run through Gradle) to make sure things are still working as they should. If any tests fail now, correct your code (either the method since it broke or the test if the test was wrong).

Your build of Dev should be successful at the end, which should show on TravisCI, take a screenshot!

Do you think your code got better? In what order would you use these quality practices in the future?

About the merge

Maybe you got a little annoyed with the merge at the end and started wondering why I did not just have you work on one branch or have them all decent from each other. It would have been easier since you would have avoided all the conflicts.

But I assume in your project you will have already learned that conflicts are hard to avoid and that sometimes work happens on the same file and everyone of you should be able to resolve conflicts.

It also hopefully shows you that in case you want to fix Checkstyle and Spotbugs violations in a large scale in your project to do that quickly and before anyone else starts working to avoid such conflicts.

Submission

On Canvas submit

1. GitHub link to your private repo (yes again)
2. Your PDF document including
    - All answers and screen shots mentioned above (sorry I am not listing them here again).