17-313: Foundations of Software Engineering

Homework 5: Quality Assurance for the People

In this assignment, you will carefully consider and engage in several QA-related processes to evaluate and look for defects in your prototype Mayan-EDMS-based graduate admissions system. The goals of this assignment are:

- Gain hands-on experience with analysis tools, including setting up, customizing, and using them.
- Practically assess and compare the costs and benefits of existing static bug-finding tools.
- Develop a plan to integrate and roll out tools in development practice.
- Explain the predictions of a Machine Learning Model, and reason about their implications.
- Build a checklist to assist you in performing code review. Put the checklist into practice by following it to perform code review on a commit in your teams' project history.

Project Context and Tasks

Quality Assurance is a critical part of software development. Although you have been testing your new graduate admissions system this whole time, you are now setting out to establish a sustained QA practice that can be used moving forward as you iterate over and continue to improve your system. Your CTO has assigned you the task of evaluating existing tools and practices beyond unit testing, and producing a report on (A) the cost/benefit tradeoffs and risks of several tools and processes and (B) how they might fit in development practice, including a mechanism for code/commit review.

Static Analysis

First, you will evaluate and choose between a set of static analysis tools, integrate it into your build/deployment pipeline, and document your decisions/process by way of a design document/RFC. For the purposes of this RFC, you must identify and experiment with at least N potential static analysis tools that are applicable to your system, where N is the size of your group. We provide a starter list below; at least one tool must be taken from either a Google search or from the Awesome Static Analysis page (that is, cannot otherwise be listed in the bulleted list).

Import the latest/best version of your code (ideally one with a large number of commits, see part (3)) from any prior homework into a fresh repository. You can create a HW 5 repository using this link:

https://classroom.github.com/g/m2U5HwLt

Apply the tools to your project. You may also apply it to one or more other programs if you wish to assess it in different contexts. Consider and experiment with the types of customization that are appropriate or necessary for this tool, both a priori (before they can be used in your project) and possibly over time. Assess the strengths and weaknesses of each tool/technique, both quantitatively and qualitatively. You might consider issues like, but not limited to: what types of problems are you hoping your tooling will catch? What types of problems does a particular tool catch? What types of customization are possible or necessary? How can/should this tool be integrated into a development process? Are there many false positives? False negatives? True positive reports about things you don't care about?

The deliverable for this part, at a high level, is a Design Document/RFC that explains and justifies the static analysis tooling you propose to incorporate into your process, and how you propose to do so. This

decision should be feature- and data-driven, and should consider usability and process questions like how and when the tooling will be applied, and by whom. See below for more details.

Starter list of Tools:

- Pylint, one of the best-known and most widely-used Python static analysis suite, provides a collection of tools, including a linter/style checker, pyreverse (which reverse-engineers UML diagrams), etc.
- Pyflakes is similar to the core linter in pylint, but with an emphasis on speed and low false positives.
- Mypy provides static type checking for Python
- sonarQube is a proprietary product targeting multiple languages, including Python; it has an open-source version you may want to try.
- Flake8 provides a wrapper around a number of other tools.
- Others are available. Awesome Static Analysis page/repo has an extensive listing of available static analysis tools for a pretty hefty list of programming languages, including Python. Other tools target Django specifically; use your Googling skills, and see what you find!

ML Model Assessment

In the last homework assignment, you created a Machine Learning model. Your CTO is seriously considering adding it into your product. However, before announcing it as a feature, she wants to ensure that your team has a deep understanding of how the ML model is working, as well as that it is tested with respect to any concerns that may have surfaced previously.

For this task, your team is tasked with writing a data-driven report for your CTO. You should evaluate the model, test outcomes, present your findings, and discuss them.

Your first task to to present a data-driven analysis of the predictions that your model is making. You will do this using the LIME tool we have previously looked at in class: https://github.com/marcotcr/lime

Run this tool on your model, and collect data on how the model is making predictions. You should use this data to report on the behavior of the ML recommendation system.

This step will allow your company to ensure that the ML is working properly. However, you are also concerned with fairness.

You should also include in your report a data-driven analysis of the fairness of your algorithm. To analyze the fairness, you should remember the fairness discussion we had in class, based on this tool: https://research.google.com/bigpicture/attacking-discrimination-in-ml/

Finally, you report should include a recommendation if you want to use the ML, scrap it, or make specific improvements before rolling it out. Specifically your report should include the following information:

- Data-driven analysis of the predictions the model is making.
- Any concerns you have about the quality of the predictions in light of this data.
- Any features in the data you are concerned about from a fairness perspective. HINT: you might want to consult your last homework when considering this.
- A data-driven analysis of the interplay between these features and your ML model. There are various ways to do this, but a simple approach might consider the following:
    - Distribution of this feature in your dataset.
    - Distribution of this feature in your accepted and rejected recommendation populations
    - Relationship between this feature and your false positives and false negatives.
- Based on this data, you should consider what is the fairness strategy that you are trying to achieve. You may use one of the fairness strategies we considered in class, or define your own. If you define a new fairness strategy, you should describe it, and present why you think it is a better fit than any of the existing strategies for this product.
- If you are not happy with the performance of the system, based on the data you have collected, you should do the following:
    - Report on what aspects of the system you are unhappy with
    - Iterate your model 1 iteration, and see if you can improve its performance. Most likely, this will NOT be enough to fix it, but your goal in this assignment is to learn enough to make a reasonable estimate of the effort needed to fix the model.
    - Make an estimate of how long it will take to bring the model up to acceptable performance. This can be a "T-Shirt" estimate (e.g., S/M/L/XL) but it should also include completion criteria. This will look like specific thresholds that your model should achieve before you would be comfortable shipping it as a part of your product.
- At the end of this report, make a recommendation to your CTO. This recommendation should be one of the following:
    - It is good enough to use now, we should ship it.
    - It is not good enough to ship, but we have a plan to improve it
    - We don't feel comfortable shipping this feature, we should scrap it.

Individual Code Review

Code or commit review is an important part of development at many modern companies. Your team realizes it's been a bit lax in this process, and resolves as part of your new commitment to QA process, to catch up.

First, to enhance review productivity, you should create a code review checklist. For this assignment, you will be (individually) creating such a checklist that will support you in performing effective code review.

A good checklist will not duplicate the issues that can be found via static analysis, but instead focuses on issues that humans are better at recognizing, such as design, readability, etc. Your checklist should also balance being thorough, but not so long that it is impractical.

Then, use that checklist as a guide to perform a commit review on a historical commit from your homework repository. To do this, *individually,* select a prior commit performed *by someone else* from your project history, and perform a lightweight code review. Ideally, everyone on your team will be able to select a different commit; contact the course staff if this is not the case.

Deadlines and Deliverables

This homework has three (2) deadlines and three (3) deliverables. The first deadline (**Tuesday, Nov 19th**) is for the *group deliverables:* the static analysis design doc, and the report on the ML model. The second deadline (**Thursday, Nov 21**) is for the *individual deliverables:* your individual code review checklist, and a code review performed on someone else's historical commit to your repository.

Part A: Static Analysis - Group (due Nov 19) - 80 points (40%)

The deliverable for this part is a Design Document/RFC that provides (1) a justified explanation for which tool(s) you think the project should use moving forward, and (2) how it shall be integrated into your process (you must recommend at least one tool, even if it's with reservations). This latter point should address both technical (e.g., at what point in the development/deployment process shall it be integrated? What sorts of customization or configuration will you be using?) and social issues (e.g., how will you incentivize the change?), as applicable. The justification should be based on your experiences running the tools and, as much as possible, be grounded in data about, for example, tool usability, output, and customizability.

Be sure the RFC also explains/justifies the alternative tools (or process options, if pertinent) that have been rejected. To receive full credit, you must consider at least N total tool options in your RFC, where N is the size of your team.

The document should also contain other relevant sections for a Design Document/RFC for this type of (development process) feature. Are there open questions? Issues you consider out of scope? Drawbacks of the proposed process/tooling that you are accepting for some (good) reason? Etc. That is: make sure it's a good/complete design document!

Submit the Design Document as a single PDF to Canvas.

Part B: ML Explainability – Group (due Nov 19) – 80 points (40%)

For this deliverable, you will be collecting data, and writing a report. The report should include the data you collect as well as your interpretation of the data.

First, you should collect data by running LIME on your ML model from the last homework. You should present the results of this as data in your report.

Then, you will interpret the data to explain why your machine learning model is making predictions. This information should include the features that provide the most predictive power.

You should also evaluate your machine learning model considering fairness issues. You will evaluate the performance of your model with a specific target fairness strategy in mind, and if you are unhappy with

the fairness of the model, you will come up with thresholds that you feel the model must meet before you would feel comfortable using it.s

Based on your findings, you should recommend one of three options. You might feel that the Model is good enough to deploy as is, you might recommend specific changes before you deploy, or you might recommend it not be deployed at all.

Submit this report as a PDF via Canvas.

Part C: Code Review – Individual (due Nov 21) – 40 points (20%)

For this deliverable, you should submit your code review checklist. It need not be lengthy; it absolutely shouldn't be more than a page. The checklist should focus on finding issues that cannot be found via static analysis.

Including in your checklist PDF a link to the commit in your repository where you performed your review. You must provide at least one comment on the commit that indicates that you have checked it against your checklist (this is to help us grade!); if you identify issues, either via your checklist or otherwise, *constructively* and *concisely* comment on those as well.

Submit your checklist, including a link to your reviewed commit, as a PDF in Canvas.