CS 239 Data Science in Software Engineering Project Part B
Build your own Software Analytics!

Team Name:
Team Members with UCLA ID:
Repository URL:

Stated Goal: In Part B, your team will design, implement, and evaluate your own Software Analytics tool. The learning objectives of Part B are (1) to **build your own software analytics** tool in order to encourage your creativity and ownership, (2) to have **hands-on learning experience** by implementing a part of the concepts that we learned in the papers discussed in the class, (3) to practice technologies learned in student tutorials for example, building **Big Software Data Analytics** by using Spark or Hadoop APIs, (4) to derive **actionable insights** about the subject software systems by applying your own analytics.

Requirements: There are **three soft requirements** that your project should satisfy:

1. You should **use one of the tool tutorials** discussed in the class. These tools include Spark, Hadoop, D3, Tableu, Eclipse JDT for AST analysis, RStudio, GitPython, etc. We encourage you to explore beyond the ASM tutorial in Part A for software instrumentation and runtime profiling.
    - Since the second half of the class is emphasizing the **use of big data processing**, it will be a real plus if you use either Spark or Hadoop to implement some part of your software analytics work flow.

2. You should **re-implement, apply, or re-invent a part of the idea in the papers discussed in the class**: for example, association rule mining (Zimmermann et al), anomaly detection using frequent itemset mining (Wasylkowski et al.), genetic programming (Weimer et al), linear regression (Nagappan and Ball), decision trees (Zimmerman et al. 2009), visualization (Zhang et al), and sampling for bug isolation (Liblit et al. or Glerum et al), etc. If you are unsure about how to meet this requirement, please discuss with the instructor. We discuss a few example ideas below.

3. Based on your experience of applying your own software analytics to subject programs of your choice, you should write up about "**actionable insights" about how the subject programs should evolve.** These recommendations could be in the form of:

    - "Which areas should SW dev teams focus their **bug fixing effort** on?"
    - "Which areas should SW dev teams **refactor**?"
    - "Which modules should SW dev teams assign more **testing resources** on?"
    - "Which modules should SW dev teams **optimize performance**, etc?"
      The recommendation should be concrete, insightful, and *ideally* actionable.

Example Project Ideas:

Here, I discuss a few example project ideas that meet the above three soft requirements. You are welcome to implement one of these ideas directly, or adapt them, or propose your own idea.

- **Idea A: Mining Frequently Called Method Patterns for Identifying Refactoring Opportunities**
    - Synopsis: Based on the work you have done in Part A on Calling Context Tree based profiling, you can identify what are the set of children methods that are called together from the same method. For example, suppose that CCT has a method node A, whose children are methods B, C, and D. In other words, these methods B, C, and D are called from the same parent method A. By representing these children nodes in terms of transaction in the association rule mining, you may be able to learn rules such as "when A is called, B and C are also likely to be called with high accuracy." Based on these rules, you may be able to find refactoring opportunities for restructuring the API of the subject programs. For example, since B and C are called frequently from A, you may be able to combine them into a new unified method X that invokes both B and C, thus making it easier for the programmer to always use B and C together.
    - The above idea meets Requirement 2 by re-implementing the association rule mining algorithm.
    - The above idea meets Requirement 3 by making recommendations on how to restructure APIs.
    - You can make the above idea A to meet Requirement 1 by using the association rule mining algorithm implementation in MLLib in Spark. https://spark.apache.org/docs/1.6.0/mllib-frequent-pattern-mining.html
    - You can also make the above idea A to meet Requirement 1 by using D3's tree visualization to visualize frequently called method patterns in CCT.
- **Idea B: Are Linguistically Complex Software Files Defect-Prone?**
    - Synopsis: You may have a hypothesis that linguistically complex modules (Java files with a large set of word vocabularies) are defect prone modules (Java files with a high frequency of bug fixes in the version history). To investigate this hypothesis, you will need to extract several source code metrics at the file level---such as Churned LOC / Total LOC, Deleted LOC / Total LOC, Filed Churns / File count, Churn Count / Files Churned, etc. similar to Nagappan and Ball's metrics. You can also extract the bug fix counts per file by searching for the commits with a commit message "fix" or "bug" using GitPython. To model linguistic complexity, you may extract a set of distinct vocabularies per file by adapting the Spark Word Count program to tokenize identifier, method, and type names in source code. For example, a method declaration with the name "getHelloHelloWorld" with a method body {print ("Hello");} will generate the set of vocabularies, { (get,1),

(Hello,3), (World,1), (print, 1)} with the total number of distinct vocabulary, 4. You can build a linear regression model to see whether "linguistic complexity" is a good predictor for defect-proneness among other metrics. Based on your study, if the linguistic complexity is correlated with defect proneness, you can make recommendations on which files developers need to focus their renaming effort in order to produce a concise and consistent vocabulary set. Or you may predict which files are likely to experience future defects due to its linguistic complexity.
    - The above idea meets Requirement 1 by using Java AST parsing to extract identifier, method, and type names.
    - The above idea meets Requirement 1 by adapting the Spark Word count program to tokenize identifier, method, and type names.
    - The above idea meets Requirement 2 by building a linear regression model for defect prediction. The idea is similar to Nagappan and Ball but uses a linguistic complexity metric based on the vocabulary count in addition to other metrics.
        - *An easier option for the above project idea is to skip extracting many different metrics and building a linear regression model. Instead, you can simply measure the Spearman correlation between the linguistic complexity and defect counts, still meeting the requirement 2.*
    - The above idea meets Requirement 3 by making recommendations on which files to apply rename refactoring or predicting which files are likely to have more bugs.
- **Idea 3. Spectra-based Fault Localization**
    - Synopsis: You can build spectra-based fault localization at the level of method calls if you have both failing vs. passing test cases. Then you can use D3 or Tableu to visualize which methods are likely to be the root cause of faults by adapting 4 different suspicious scores mentioned in Zhang et al. to the level of method calls. You can make recommendations on which method calls are likely to be the cause of test failures.
    - The above idea meets Requirement 1 by using D3 or Tableu visualization tool kit.
    - The above idea meets Requirement 2 by adapting 4 different suspicious scores to work at the level of method calls.
    - The above idea meets Requirement 3 by making recommendations on what could be likely causes of faults.
    - The challenge of this project is to collect enough tests so that you have a good number of failing and passing test cases to draw meaningful conclusions.

Due Date: March 7th 2016, 12AM.
Presentation Dates: March 7th and March 9th in Class
You must submit both report and presentations before March 7th 2016, 12AM. However, in order to allow more time for implementation, you will be allowed to update your report until **March 14th, 2016, 12AM, which is a hard deadline**. The class presentation about your projects will occur on Monday 3/7 and Wednesday 3/9 of Week 10.

Report Format: 11pt document. Approximately 7 pages in Word or PDF
Presentation Format: 10 to 15 slides in Powerpoint or PDF