Student Name 1 : Adavya Bhalla
NetID : bhalla
Student Name 2 : Satvik Agarwal
NetID : satvik22
CSE 403 Project Pitch

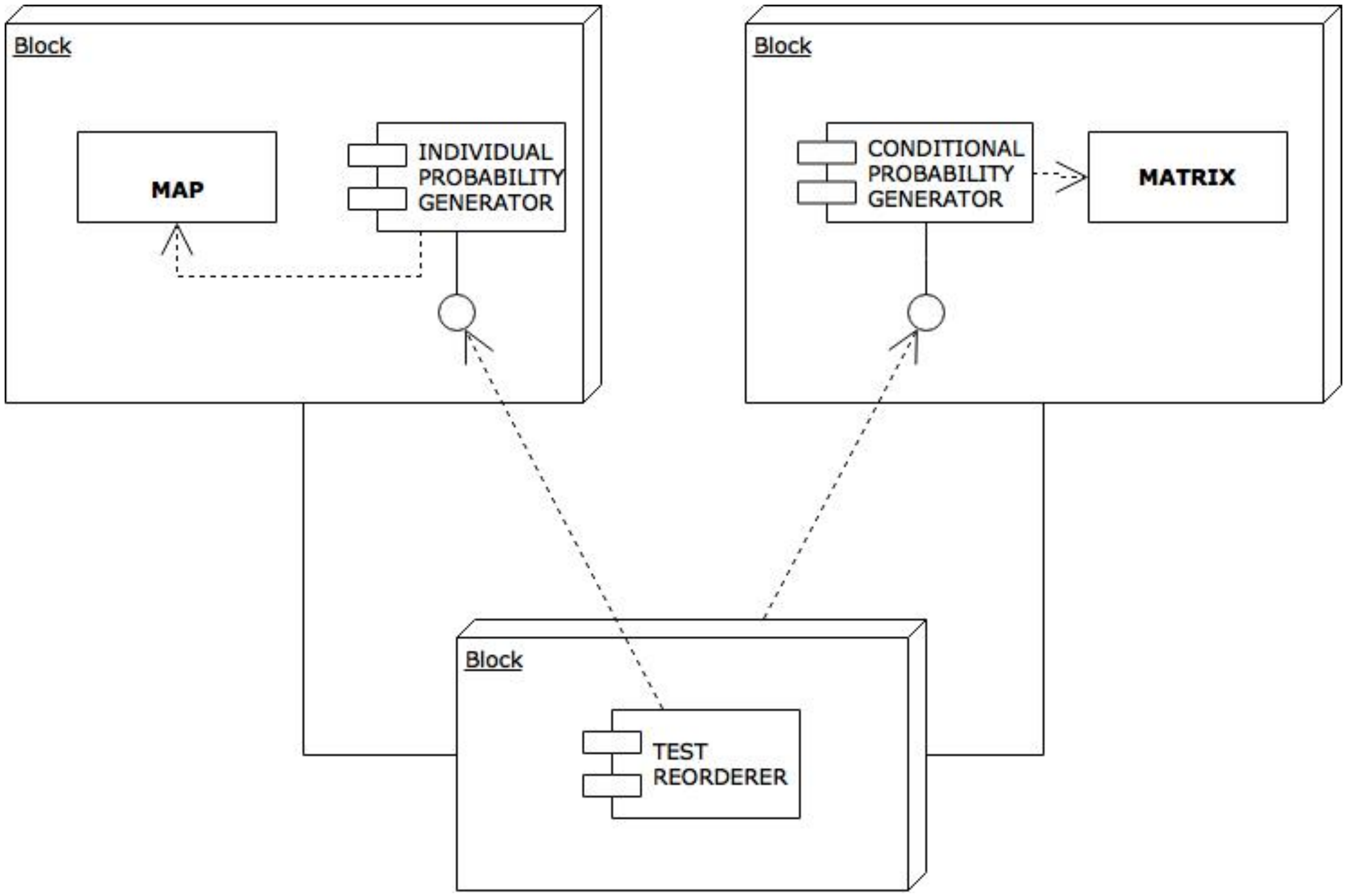
TEST BAYES!

Motivation : Software applications nowadays have become extremely complicated and difficult to test and verify. This can be attributed to the fact that the problems we are solving have become increasingly complex and developers cannot afford to make a product that is not technically sound or something that has not been exhaustively verified and tested. More often than not, testing takes a lot more time and effort than the actual solution to the problem. If we can devise a tool that reduces the amount of time it takes to test/verify a given solution/implementation of a solution to a problem, then that would be a major boost to the current development and testing process. Moreover, the number of tests nowadays are also substantial in number and when we run a test suite, we expect it to indicate results quickly. However, if a test suite is very large and is ordered in a way such that the first half or more passes but the others fail, then the developer loses out on precious time as the tests could have been ordered in a way particular order to indicate errors first and fail fast. In such a scenario, the developer is likely to save a lot of time and come up with a solution quicker since
the developer was just working on the code. Besides, if the tests that failed had an intelligent ordering i.e. debugging the tests that failed would lead to a lot of errors being discovered in contrast to multiple failed tests indicating the same error, then the job of the developer is made even easier since the tests that failed initially can help the developer discover a lot of bugs in different areas of the project quicker.

Approach : The re-ordering of tests can be treated as a probabilistic problem where we take advantage of not only the independent probability of each test failing but also the co-related probability of that test conditioned on other tests. The probabilistic treatment ensures two things : 1) Tests that usually fail after an update are tested first and 2) The ordering is intelligent in the sense that these tests indicate possibly different bugs. This happens because the tests reorder themselves at run-time. Tests are reordered based on the outcome of the previous tests. This ensures that the order is interesting i.e. it is more likely to fail fast and indicate diverse errors. To achieve this goal, we can have the following high level approach :

1) We maintain a list of the independent probabilities of all tests failing on different versions of the same project. This list is constructed as we run the test suite on different versions of the implementation. For example, to construct the list of probabilities, we go through 100 different versions of the same large project and find out the probabilities of individual tests failing.
2) Next, we have to construct multiple data structures that store all the conditional probabilities of these tests. If there are n tests then we can implement this using two n*n matrices. The first matrix stores the probability of a test failing given that another test succeeded while the second matrix stores the probability of a test failing given that another test fails.
3) Finally, we first run a test that has the highest probability of failing irrespective of other tests. If this test fails, then we iterate through the second matrix and find out which test has the highest probability of failing given that the first test failed. If it passes, then we do the same thing with the first matrix.

Challenges and Risks : It is difficult to mathematically verify whether this approach will be successful or not. It sounds correct but may not be technically sound. The biggest risk is that the overhead from running this extension tool might take longer because of the initial training and construction of the two matrices and the list. Also, it might be challenging to find such a large open-source project that has so many previous versions.
Here is a small diagram that illustrates this procedure :


Block
MAP
INDIVIDUAL PROBABILITY GENERATOR
Block
CONDITIONAL PROBABILITY GENERATOR
MATRIX
Block
TEST REORDERER