Project Progress Review #4

(Test Plan)

Project Title	:	Efficient Python Genetic Algorithm Framework	
Project ID	:	PW19CGM01	
Project Guide	:	Ms. Chitra G.M.	
Project Team	:	Bharatraj S Telkar	(01FB15ECS066)
		Daniel I	(01FB15ECS086)
		Shreyas Vivek Patil	(01FB15ECS286)

Project Background Details

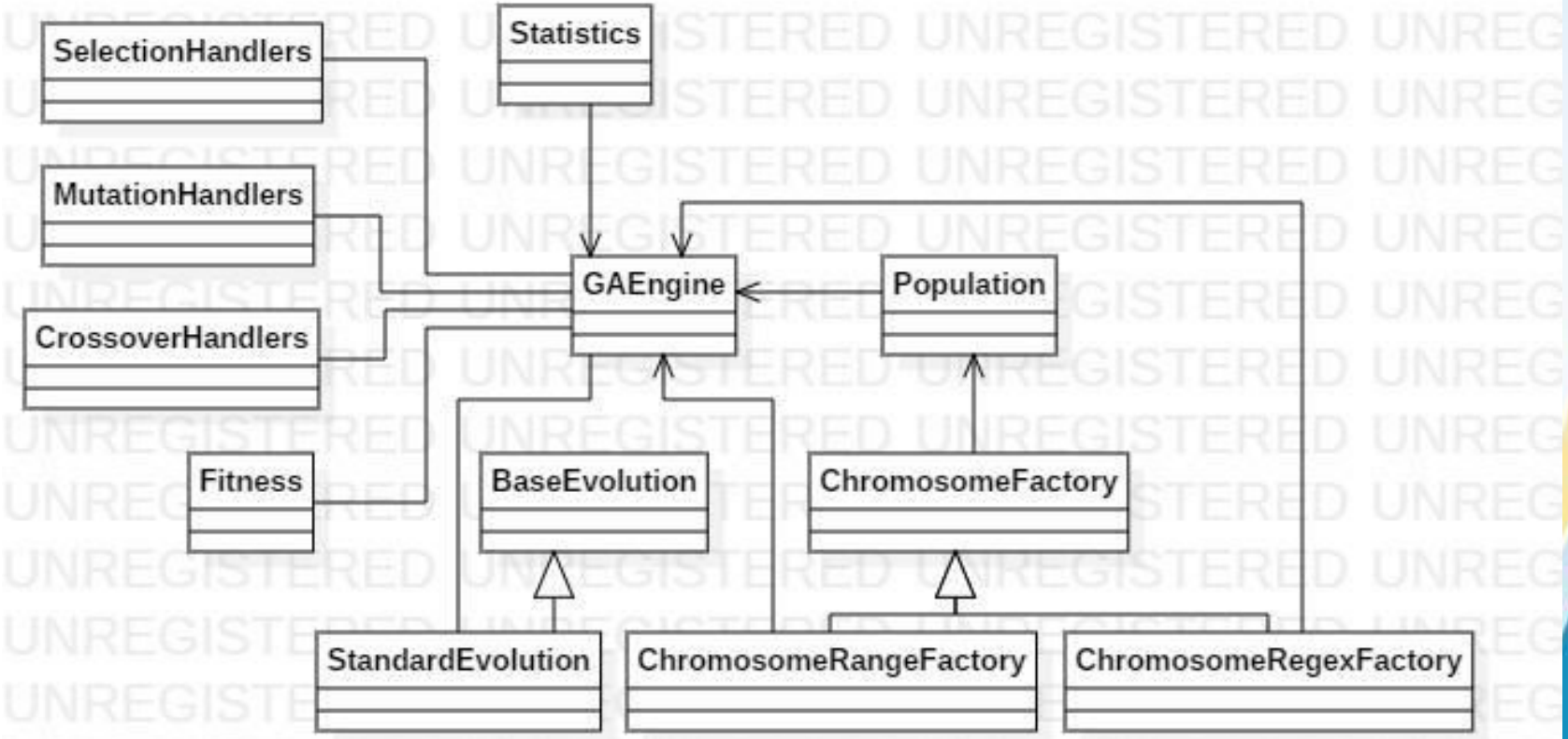
- Developing a generic API which helps users develop **a variety of genetic algorithms in a easy simple manner**

- **Usage of Spark improve efficiency of GA** (parallelisation of fitness calculation, selection, mutation, crossover. Can use large populations)

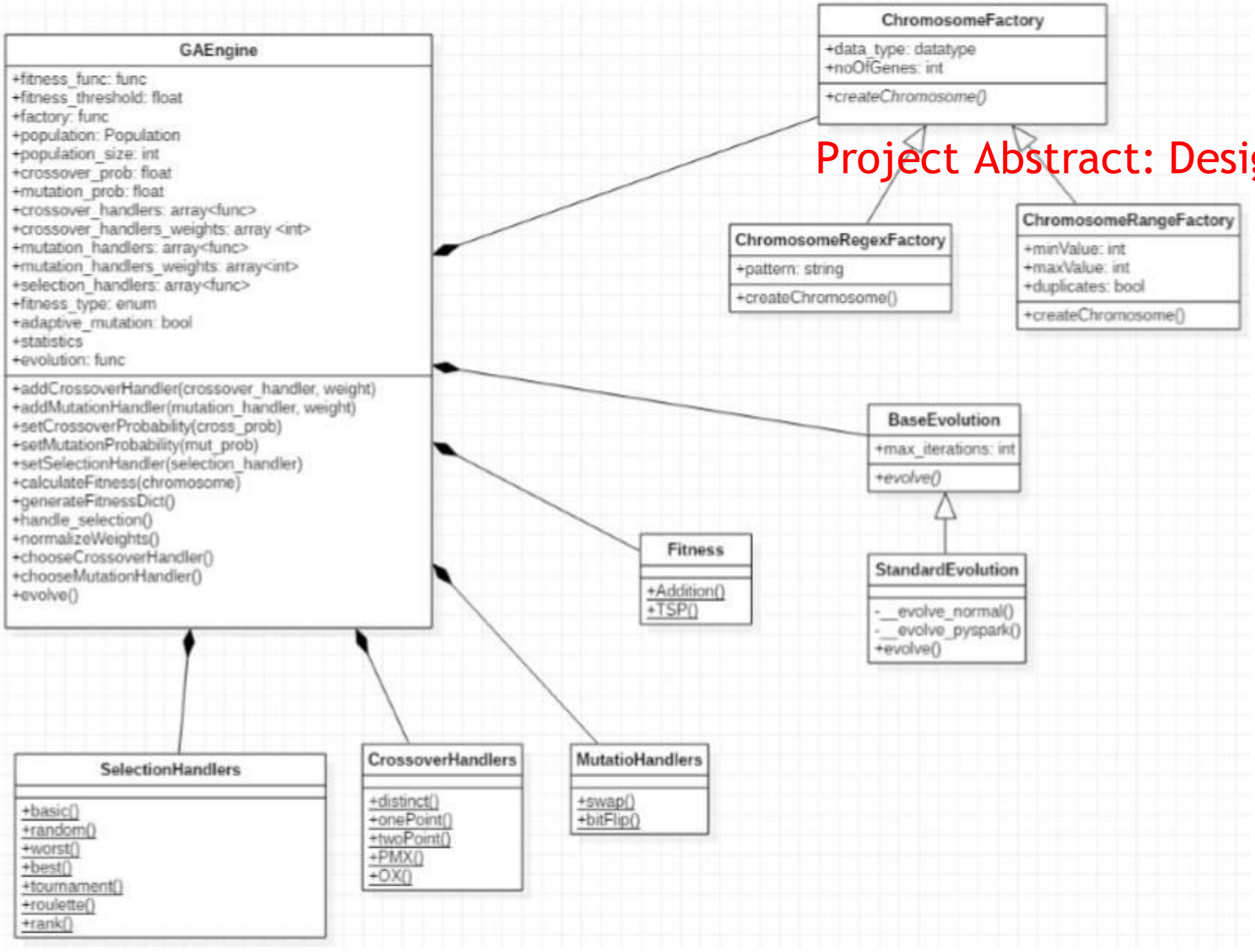
Project Background Details

- Implementation of GA optimisations including **adaptive mutation, injection of hall of fame chromosome, iteration early halting**
- ANN Evolution using GAs – to decide topology
- Developing a user friendly web site where users can simulate GA online

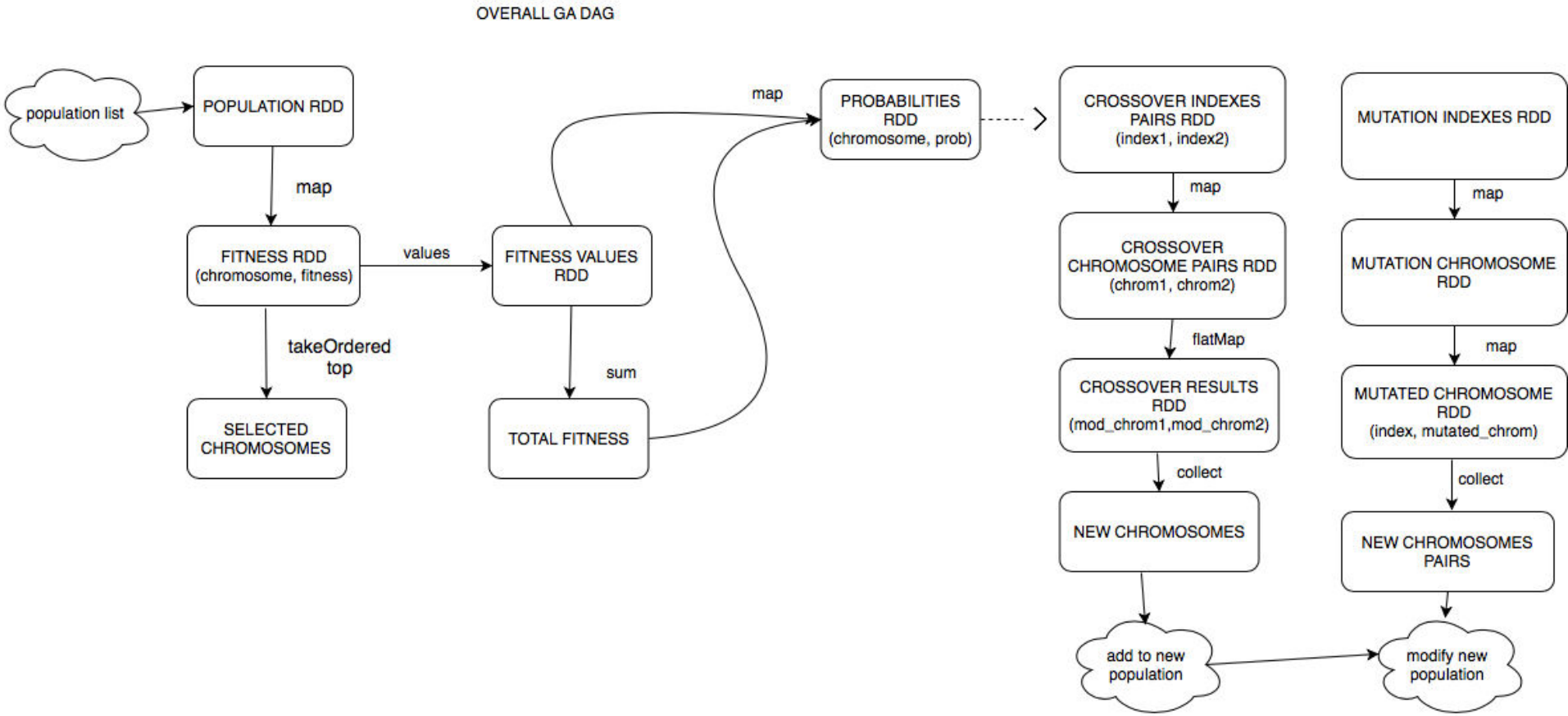
Project Abstract: Design



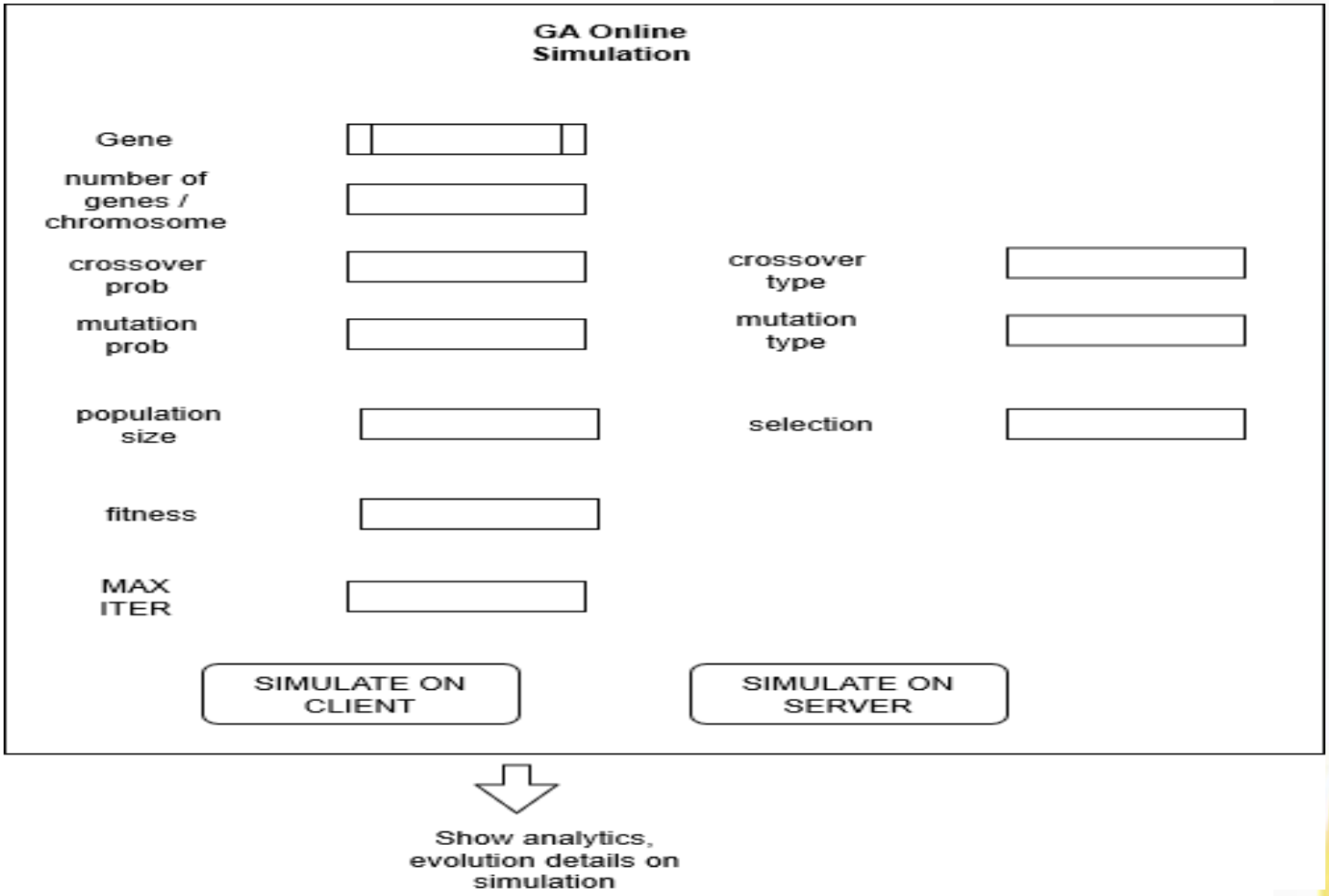
Project Abstract: Design



Project Abstract: Spark DAG

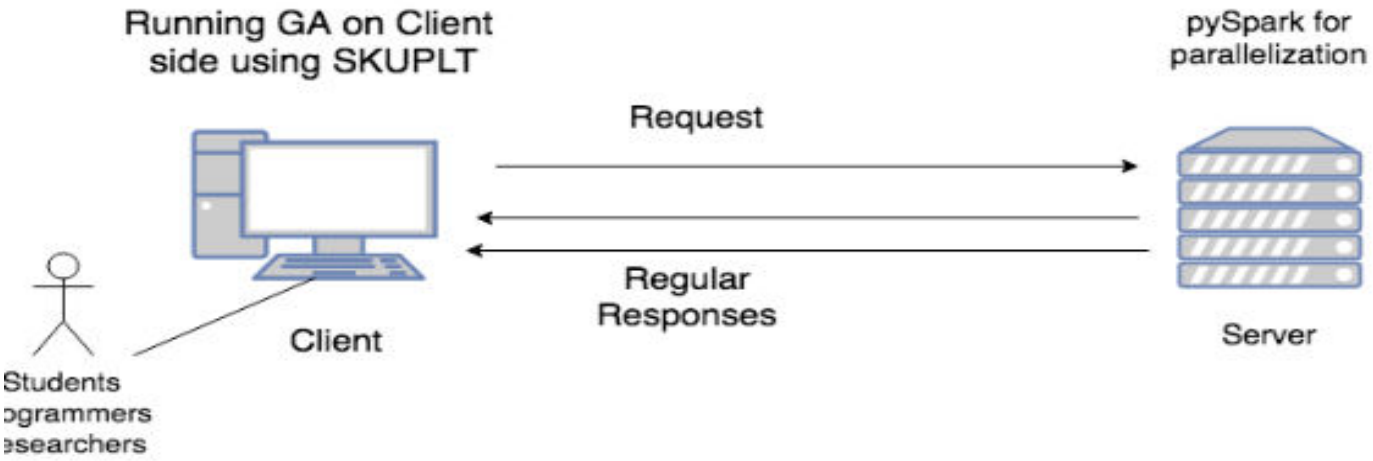


Project Abstract: UI Design

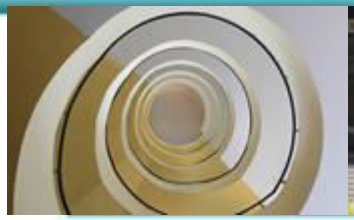


Project Abstract: Website Design

Step 1: Take users inputs about the GA from the UI
Step 2: Convert the user inputs into python code which uses our GA API
Step 3: Run the code and observe GA execution and results



PES UNIVERSITY






Test Plan

PES UNIVERSITY




1. Scope of Testing

Scope

- **Testing every possible part of our framework is infeasible** due to time and resource constraints.

- Scope of our testing can be determined by
- Those features which are the **most important features** of the project
- Identification of **features whose failures are catastrophic** to the project
- Those features which are **complex** in nature

Scope

On these grounds, the features to be included in our testing include

- Testing the **functionalities of classes** of the GA.

- Testing if the **performance of the Genetic Algorithm** is acceptable or not. (avg time taken, memory used, number of failures). Need for some acceptance criteria.

- Testing the **security of the application**. no invalid inputs are fed into the API.

- Testing the **usability of our website**.

Scope

- Testing the **documentation of our API** to ensure that every end user of our API can effectively know how to use our API.
- Testing the **conformance of the object oriented code to SOLID principles** to ensure high maintainability.
- Testing the GA website **performance under heavy loads**.

Scope

On these grounds, the features which are cautiously not being included for testing is

- Cross browser testing of our website
- UI tests which involve checking whether **certain css styles are applied when an event occurs** (eg: red color for bad input) or **DOM elements are added when an event occurs.** These can be checked manually and we need not waste time coding test scripts.

2. Test Strategies

Test Strategies

1. White Box Testing: Testing all classes of GA API

a. **Static Checks:** need to be done as soon as a module of code is developed

Manually examine each class of our GA API to find
- errors in GA code
- further possible optimisations
- check error handling capabilities of our GA API

Use of **static analysis tools pylint** to detect static coding errors in API

Test Strategies

1. White Box Testing: Testing all classes of GA API

b. Structural Testing:

- Functional Testing: Test functionalities of all functions of every class of the GA API by testing it with a set of given inputs against expected outputs.

- Code Coverage: to ensure that a higher percentage of the GA framework code is covered by the test cases.

All can be easily done using **pytest**.




2. Black Box Testing: Testing all end use functionality of all classes

a. Requirement Based Testing:

Test all the main requirements of the GA classes without knowledge of the internal data structures used

Eg: Test if Regex Chromosome Factory fulfills its requirement of generating chromosomes based on the given regex




2. Black Box Testing: Testing all end use functionality of all classes

b. Positive/Negative Testing:

Test that the code works under positive and negative conditions and doesn't crash under any conditions. (Invalid input data/ user behavior)

Eg: Normal code sequence: ga = ….
ga.evolve(10)
ga.continue_evolve(5)

Abnormal Code sequence: ga = ….
ga.continue_evolve(5)




2. Black Box Testing: Testing all end use functionality of all classes

c. Boundary Value Analysis:

Analyze if code works as intended given boundary values of input variables.

Eg: GAEngine even works with 1 gene in chromosomes in the population provided there are no crossover handlers
Population = 1




2. Black Box Testing: Testing all end use functionality of all classes

d. User Documentation Testing:

Our documentation which is hosted on readthedocs.org needs to tested through multiple reviews and formal inspection

GA Code Documentation hosted on readthedocs.org needs to be regularly reviewed by project owners




3. Error Handling Tests

To ensure that all possible error conditions are handled by rasing valid exceptions

Eg: 1. Invalid Regex given to regex chromosome factory
2. Negative population size
3. max value less than minvalue in range factory




4. GUI Tests

Usage of selenium

- To verify form validation tests – form does not support on invalid input conditions
- To verify Evolution details come to the UI on selecting “RUN GA” button
- Verify the time of receiving a response from the Server is acceptable or not




5. System Tests

a. Performance/Load testing: is to be done to evaluate the average time taken by the Genetic Algorithm to solve various types of problems either using Spark or through normal execution and to verify if it is acceptable or not.




6. System Tests

b. Stress Testing: to ensure that the GA website will not break when resource limits like disk space, memory, processor utilization have been exceeded.




6. System Tests

c. Concurrent Testing: detect the defects in the application when multiple users are logged in

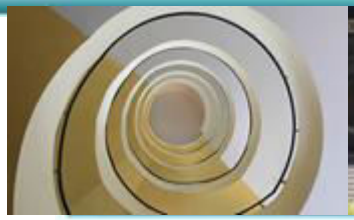
All these System Level Tests can be achieved using Locust – A python based performance testing tool




7. Acceptance Tests

a. User Friendliness Testing: Feedback from many people that it is easy to use.
b. Alpha/Beta Testing: functionality verification, etc from end users

PES UNIVERSITY






3. Test Environment

Test Environment

OS: Ubuntu 16.04
Hardware: Linux system with a fast processor(2GHz) and large memory.
Software: Scala, Java, Apache Spark, Python3, python modules defined in the requirements.txt.




4. Risks




Risks

Risk #	Risk	Nature of Impact	Contingency Plan
1	Insufficient time for testing	Defects reach customers, chaos/pressure among the testing team	Distribute testing through product life cycle, automated testing activities, have upfront test schedule
2	Development delay	Less time for testing, delay in final product	Involve constant testing in development plans, V model testing approach
3	Unclear requirements	Need for rework User dissatisfaction	Approval cycles for various requirements
4	Over cautious in testing	Wasted time, delayed product release	Setting objective exit criteria for testing
5	Lack of experience in testing	Poor testing quality	Constant skill upgradation through elearning testing techniques

5. Roles and Responsibilities

Roles and Responsibilities

Identification of roles and responsibilities of all parties involved in testing

Bharatraj, Daniel and Shreyas to work parallelly on different testing tasks as when different modules are developed

A selected 10 classmates – Beta Testing

6. Test Schedule

Test Schedule

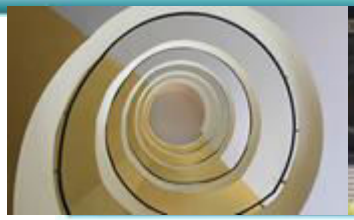
This section shall describe the schedule planned for testing activities.

Feb – 10 March: Basic Class Design, Test case generation

11 March – 30 March: White and Black Box Testing

1 April – 15 April: Performance Testing, System/Acceptance Testing

PES UNIVERSITY






7. Test Tools

Test Tools

1. pyTest: for white/black box testing

Testing all functionalities of classes, error handling capability
Can be used to test all functionalities including pySpark tests, evolution.




2. Unittest.mock: Used for various mock tests

Since most of our functions involve random values, mock.patch helps us set random() to return a determined value, prevent calling unneeded code.




3. coverage: Used for finding code coverage percent in test runs.

Pytest has a coverage option downloadable.





4. Pylint:

Pylint: For static python code testing

5. TravisCI:

Used For Continuous Integration




6. Locust:

Used For various performance testing

- Find average response times
- Performance with concurrent users
- When system breaks




7. Selenium

GUI Testing

- To verify form validation tests – form does not support on invalid input conditions
- To verify Evolution details come to the UI on selecting “RUN GA” button
- Verify the time of receiving a response from the Server is acceptable or not
- Verify that injection attacks cannot take place from GUI

8. Acceptance Criteria

Acceptance Criteria

- The website system should be able to support as many as 100 concurrent users with average response times of not more than 5 seconds.
- The website system should be very user friendly.
- The GA framework should be highly efficiently by exploiting pySpark parallelization. It should solve the GA atleast three times faster than sequential execution for large population sizes.
- The API should be very well documented.

9. Test Cases List

Test case list

A list of different test cases can be found in the Test Plan document.

Document Walkthrough

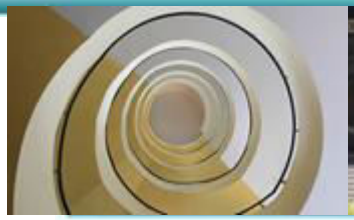
10. Test Data

Test Data

Test Data includes set of Genetic Algorithms population with evolution over many generations with a given mutation rate, etc. Random values are mocked. Using this evolution can be verified, Test Data of Chromosome Factory and some expected chromosomes and other input output data pairs using which the tests are performed.

Mock tests need to be run to test functions which involve randomness like crossover, mutation, etc

PES UNIVERSITY





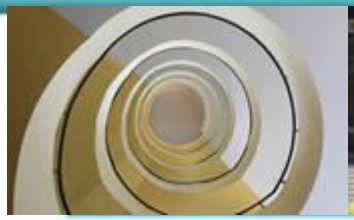

Project Progress so far

Project Progress so far

All Utils, Optimizations, pySpark parallelization feature, ANN Evolution using GAs and High level api done
All documentations complete
Website complete

Need to complete code documentation on readthedocs.
Need to document various performances – pySpark vs without pySpark, with pySpark (different workers)

PES UNIVERSITY






Thank You