*University of St. Thomas*

*Graduate Programs in Software*

*SEIS 610: Software Engineering Summer 2015*

Evolutionary Computing System

Project Plan

**06/17/2015**

hughbert kumwesiga

lu gan

mohamed hassan

meenakshi cj

xiaozhao song

anne maki

**Contents**

1. Project Description
2. Requirement Analysis
3. Data Gathering
4. User Interface / Front-End Requirements
5. Audience/Users
6. Design
   1. Data Model
   2. System Architecture
   3. Class Diagram (OO diagrams)
   4. Data flow diagram
   5. Data Structure
7. Implementation
8. Testing
9. Work Breakdown/Timeline
10. Weekly SCM
11. Project Description

The main purpose of this class project is to develop a evolutionary computing (EC) system in which a population of computer programs can automatically evolve through several generations based on the learning environment acquired (training data) by the system, generating a best candidate solution that is equivalent to the target mathematical function predefined as **y = (x2- 1) / 2.** The program structure used in this project is binary tree structure assembled by arithmetic functions (plus, minus, multiply, and divide) and terminals (integers 1 through 9, and variable x).

Inspired by Darwinian principles, the proposed EC system employs analogies to evolutionary genetics and uses genetic operations to simulate an evolutionary process for solution optimization. In other words, this EC system involves crossover, mutation, and reproduction as genetic operators to generate new variants (solutions) to be selected for the best candidate solution, driven by fitness-based selection. Such an EC system introduces an increased level of flexibility and reusability, and is of ultra importance in meeting requirements from the ever-changing reality.

This proposed EC system follows a generational algorithm:

1) Initializing a population

2) Evaluating each individual programs in the existing population (done against training data) and assign a fitness value to each individual;

3) Repeating the follow steps until the new population is fully populated:

a) Selection of individuals based on project settings;

b) Performing genetic operations (crossover, mutations, and reproduction) on the selected individuals;

c) Insertion of the result of the genetic operations into the new population;

4) Iterating steps 2 and 3 until the optimal or near optimal individual is found;

5) Presenting the best individual in the population.

In this class project, a quantitative post-project analysis of the system will be also included to report the structural and component complexity, logical complexity, performance monitor, and etc.

1. Requirement Analysis:
   1. Goals

* Design and build an Evolutionary Computing System using current Software Engineering practices.
* The system should provide a function that closely matches the target function, y = (x2- 1) / 2.
  1. Functionality

The system must use genetic programming to find the closest matching function for a given target function. Genetic programming is a method that genetically evolves a population of solutions to solve a problem. It iteratively transforms a population of solutions into a new generation of solutions by applying genetic operations. The genetic operations include crossover, mutation and reproduction. The five major preparatory steps for genetic programming require the human users to specify

* the set of terminals (e.g., the independent variables of the problem, zero-argument functions, and random constants) for each branch of the to-be-evolved program
* the set of primitive functions for each branch of the to-be-evolved program(e.g., + , - , \* etc.)
* the fitness measure/threshold
* certain parameters for controlling the run, and
* the termination criterion and method for designating the result of the run
  1. Scope

The random functions are represented as binary trees having the following mathematical operators or operands.

* Operators: +, -, \*, /
* Operands: 0,1,2,3,4,5,6,7,8,9 and x
  1. Constraints
* The total execution time for the EC system to find the best function should not be more than 15 minutes.

1. Data Gathering
   1. Training data

* The training data we are going to use is about 50-100 sets of values (x1, y1; x2, y2; …… xn, yn)
* The range for x values will be large and includes both negative and positive numbers.
* The training set will be stored in text file.
* The values of x and y are real numbers.
  1. Settings data

* Probabilities for genetic operations are
  + Crossover probability
  + Mutation probability
  + Survival probability
* Population size is tentatively chosen as~ 100
* Maximum Tree height
* Fitness margin

1. User Interface / Front-End Requirements

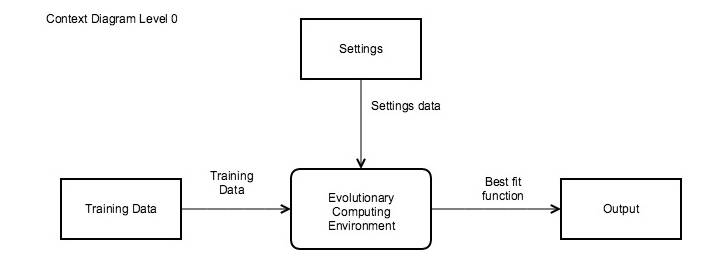
* User will have limited interaction with the system
* The following data will be displayed for the user:
  + Final function generated by the E.C. System
  + Elapsed Time
  + Graph Output showing the target function (real output) and equivalent function (expected output)
  + Fitness Value

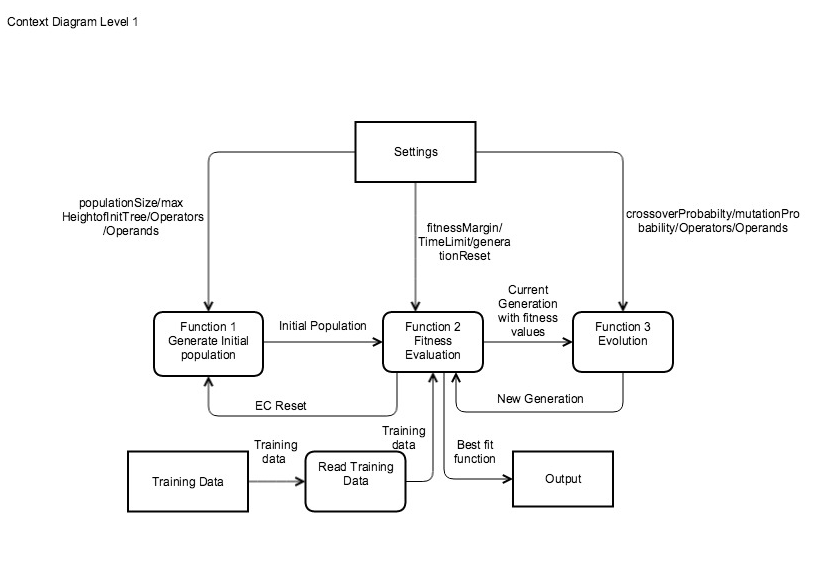
1. Audience/Users

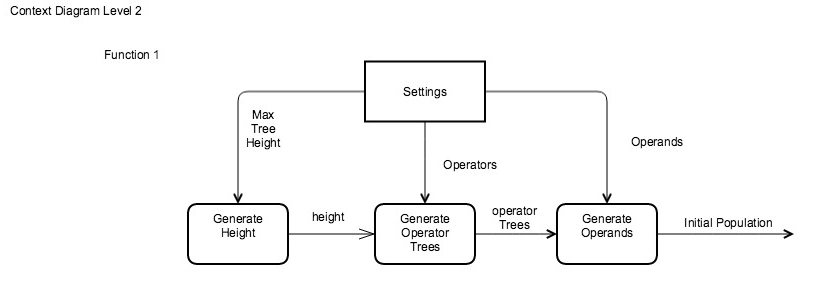
* The target audience is any user interested in finding a function that closely matches the target function based on the training data provide by the user.

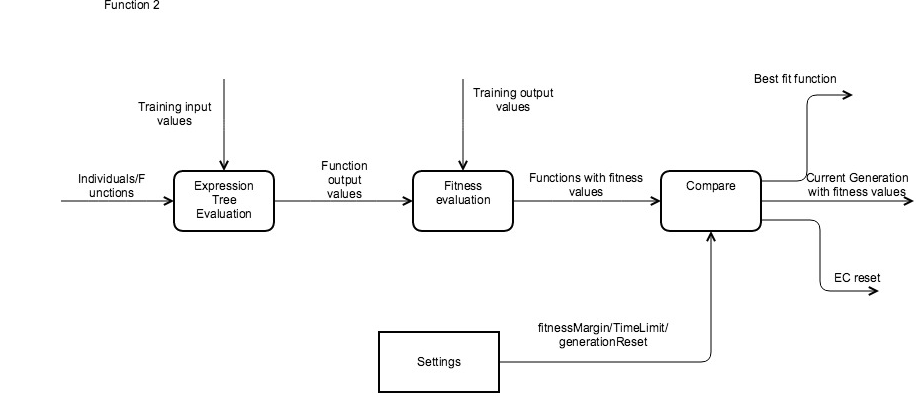
1. Design
   1. **Data model**

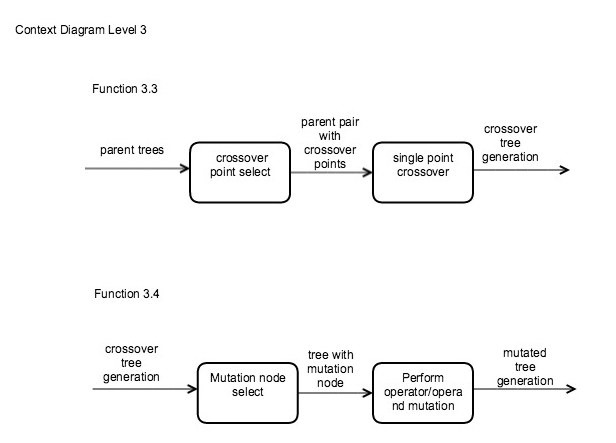
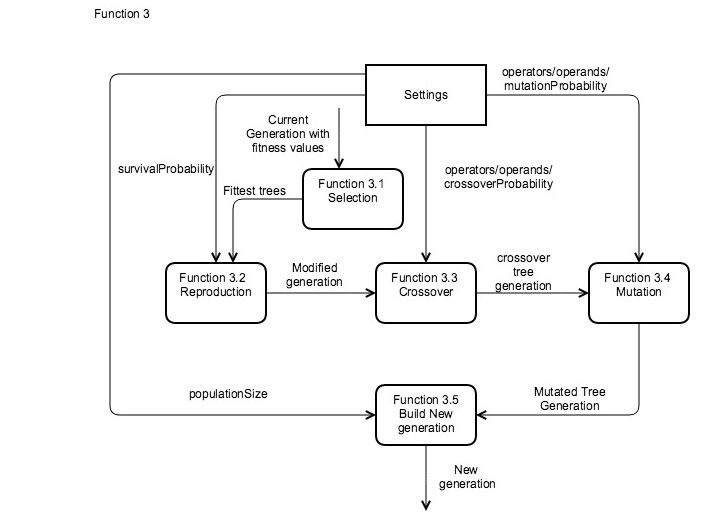
* Object oriented model is chosen for our system design. The following are the reasons for selecting OO approach :
  + Use Polymorphism
  + Easier to make modifications to the code
  + Reusability
  + Open Recursion
  + Extensibility
  1. **System Architecture**
* We will build the system from the ground up to have better control and understanding of the system. For example, developing the system from scratch reinforces the concepts that we’ve learned in class. Also, writing the code from scratch lends to better quality control. Flexibility is also another advantage of building the system from the ground up.
* We will be developing the system using incremental agile methodology to discover any potential gaps after development of each function.
  1. Data Flow diagram



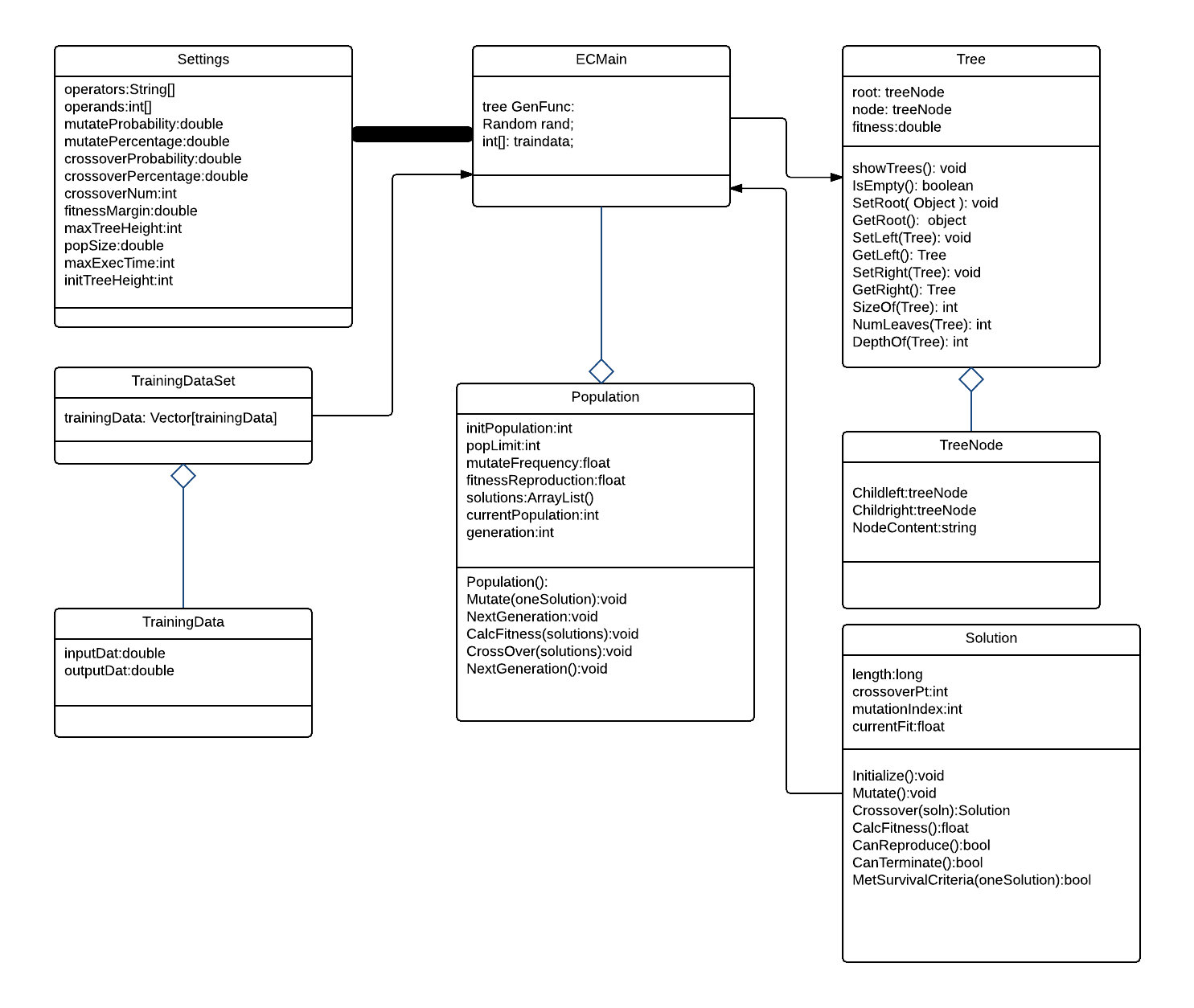








* 1. Class Diagram (OO Diagram)



* 1. Data Structure

We have used binary trees with nodes that contain operator and operand data. In addition to the built in properties of the binary tree, a genetic programming tree has the fitness property. We choose to use the binary tree data structure because of the flexibility and recursively traversing, performing genetic operations of crossover and mutation, and in the evaluation of the algebraic expression.

The Java ArrayList was our choice to represent a population of Genetic Programming trees, we used the structure as it allow us to easily do the following:

* Iterate over the population and access it using arraylist indexes
* Its sorting capabilities for population of trees

7. Implementation

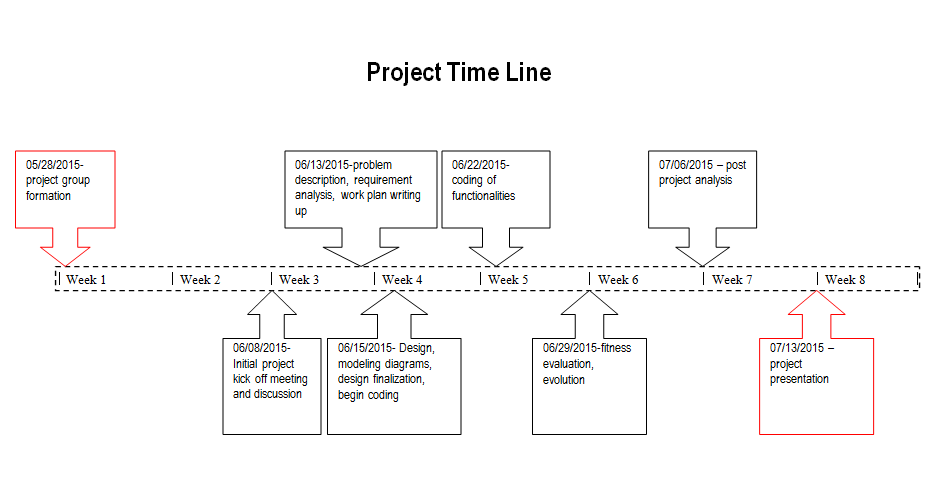
* We have selected Java to implement the EC system
* We will use JFreeChart with in Java for function graphs
  + URL: jfree.org/jfreechart
* We are using GitHub to host all the application source code, documents, Test data, and test results. Using repository in GitHub will give out the flexibility to work from anywhere and track what has being done, when, and by whom.
  + Our GitHub repository is: https://github.com/amblucas/EvolutionaryComputingProject

1. Testing

Testing will be done alongside development. The test plan will include unit tests and functional tests.

1. Work Breakdown/Timeline

* Description of Project – *All*
* Requirement analysis - *All*
* Design – *Mohamed Hassan*
* Development/Coding – *Anne Maki; Hughbert Kumwesiga, Mohamed*
* Testing - *Meenakshi CJ*
* Post-Project Analysis – *Lu Gan; Xiaozhao Song*



1. Weekly SCM

