|  |  |
| --- | --- |
|  | **2010** |
|  | **Team Tommy**  Matt Wolford  Cory Allen  Ronnie Jandou  Scott Smereka |

|  |
| --- |
| **[Software fault predictor ]** |
|  |

# Table of Contents

1. **Introduction** 
   1. Goals and Objectives Page 1
   2. Statement of Scope Page 1
   3. Software Context Page 1
   4. Major Constraints Page 1
2. **Usage Scenario**
   1. Use Profiles Page 2
   2. Use Cases Page 2
   3. Special Usage Considerations Page 3
3. **Data Model and Description**
   1. Data Description Page 3
      1. Data Objects Page 3
      2. Relationships Page 3
      3. Complete Data Model Page 4
      4. Data Dictionary Page 4
4. Functional Model and Description
   1. Description for Function: User-Input Validation Page 5
      1. Processing Narrative (PSPEC): User-Input Validation Page 5
      2. User-Input Validation Flow Diagram Page 5
      3. User-Input Validation Interface Description Page 6
      4. User-Input Validation Performance Issues Page 6
      5. User-Input Validation Design Constraints Page 6
   2. Description for Function: Predict Number of Faults Page 6
      1. Processing Narrative (PSPEC): Predict Number of Faults Page 6
      2. Predict Number of Faults Flow Diagram Page 7
      3. Predict Number of Faults Interface Description Page 7
      4. Predict Number of Faults Performance Issues Page 7
      5. Predict Number of Faults Design Constraints Page 7
   3. Description for Function: Compute Statistics Page 7
      1. Processing Narrative (PSPEC): Compute Statistics Page 7
      2. Compute Statistics Flow Diagram Page 7
      3. Compute Statistics Interface Description Page 8
      4. Compute Statistics Performance Issues Page 8
      5. Compute Statistics Design Constraints Page 8
   4. Software Interface Description Page 8
      1. External Machine Interfaces Page 8
      2. External System Interfaces Page 8
      3. Human Interfaces Page 8
   5. Control Flow Description Page 10
5. Behavioral Model and Description
   1. Description for Software Behavior Page 11
      1. Events Page 11
      2. States Page 11
   2. State Transition Diagrams Page 12
   3. Control Specification (CSPEC) Page 12
6. Restrictions, Limitations, and Constraints
   1. Time Limits Page 13
   2. Budgetary Constraints Page 13
   3. Hardware Constraints Page 13
   4. Memory Constraints Page 13
7. Validation Criteria
   1. Classes of Tests Page 13
   2. Expected Software Response Page 14
   3. Performance Bounds Page 14

# 1.0 Introduction

## 1.1 Goals and objectives

Overall goal and objective is to design and implement a simple object-oriented data mining tool which uses analogy-based reasoning to predict the number of faults in program modules.

## 1.2 Statement of scope

The object-oriented software application will use analogy-based reasoning theories to predict the number of faults for all modules in a test data file. The software will also possess functionality to exporting of an excel file that will contain computations of mean-squared-error, standard deviation, and other basic statistics for the predicted number of faults given the actual number of faults.

An upload of case library will be necessary to predict number of faults for program modules with unknown faults and known metrics,

* All modules in the case library are represented by a given number of software complexity metrics and a software fault attribute. Using the row-column format, a given module, i, is composed of software metrics values, and a software faults value {*xi1*, *xi2*, *xi3* … *xik*, *yi*}, where k is the number of software metrics, *xab* is the value in the *a*th row and *b*th column.

The upload of test data containing program modules for which the number of faults are to be predicted.

The Euclidean distance similarity function is required to compute the distance between modules i and j by a mathematical approach that determines the similarity factor between two program modules

* *dik =* SQRT{ Σm*k=1* {*xik - cjk*) 2 }

The solution algorithm will be used to predict the number of faults in a program module based on a set of nearest neighbors (a user-defined number of most similar modules from the case library) through a mathematical approach.

* *p\_yi* = [Σ1nN { *yi* }] / (*nN*)

Where, *nN* is the number of nearest neighbors to be used for prediction, and *p\_yi* is the predicted number of faults values for the *ith* module.

## 1.3 Software context

The fault predicting software will be used by software developing companies for the means of improvement in software quality and software testing process.

## 1.4 Major constraints

The software will require a user-friendly GUI, as a different range of users will be using it, and the software must be object-oriented.

# 2.0 Usage Scenario

## 2.1 User profiles

The fault predicting software will be intended to quality assurance groups, software programmers, software engineers, and project managers, all the same level user. There is no distinguish between different types of users that will be using the software; they will all have the same functionality and access.

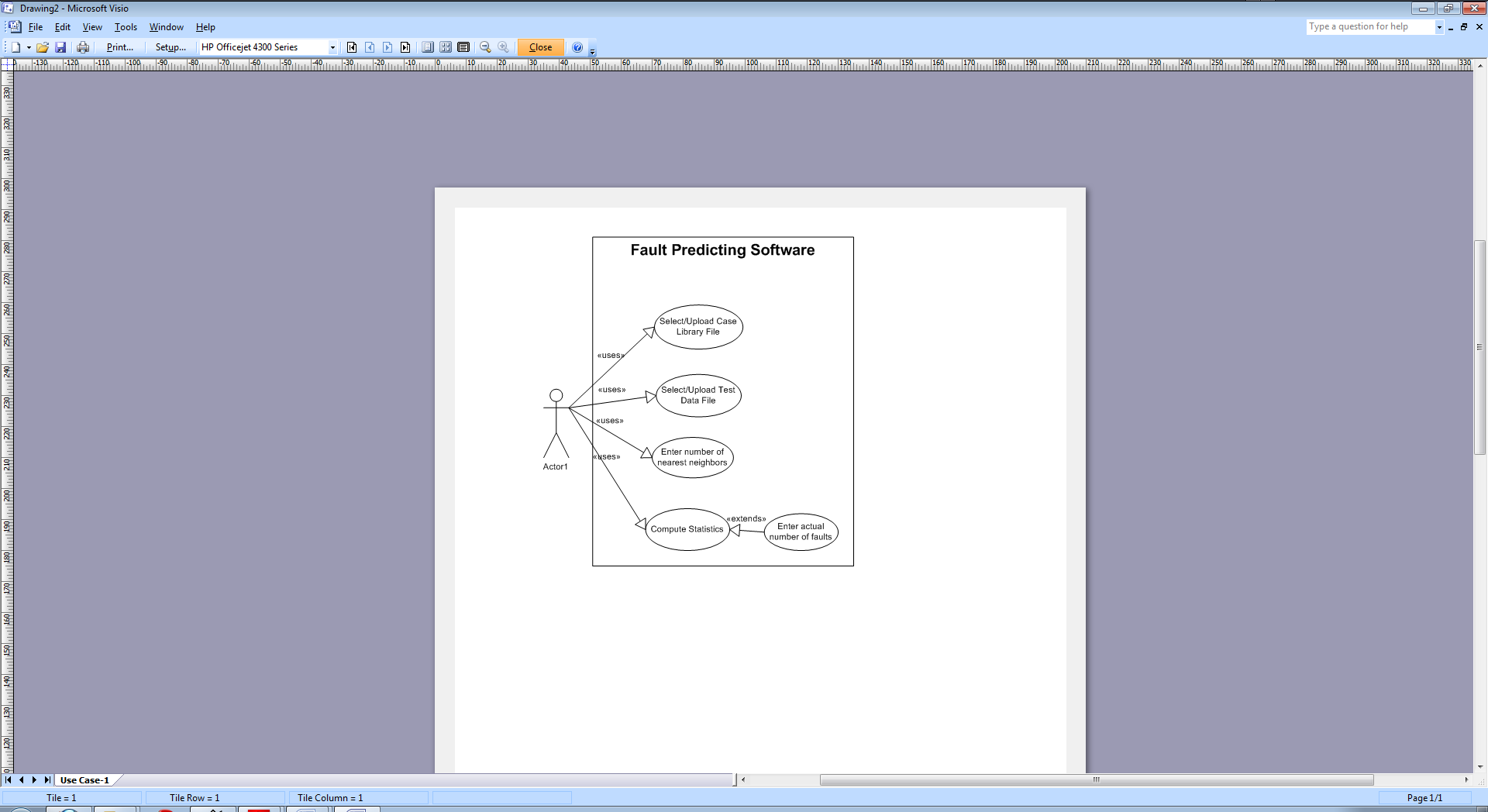
## 2.2 Use-cases

***Figure 2.2-1 Fault predicting* as a very high-level use case**

Fault predicting

* User selects and uploads a case library file.
* User selects and uploads a test data file containing program modules for which number of faults are to be predicted.
* User enters the number of nearest neighbors to be used for prediction.
* User requests the tool to obtain the predicted number of faults for all modules in the test data file.
* Software calculates the predicted number of faults for all modules in the test data file.
* User selects and uploads (if known) the actual number of faults for the test data file modules
* System computes mean-squared-error, standard deviation, and other basic statistics for the predicted number of faults given the actual number of faults.

**Figure 2.2-2 *Fault predicting* as a use-case diagram**



**2.3 Special usage considerations**

The uploaded case library files must be properly formatted using a space delimited and rows-columns format in a flat text file. The software uses the simple average to compute the predicted number of faults; however, if the users are creative, they may allow an option of choosing the median instead.

**3.0 Data Model and Description**

**3.1 Data Description**

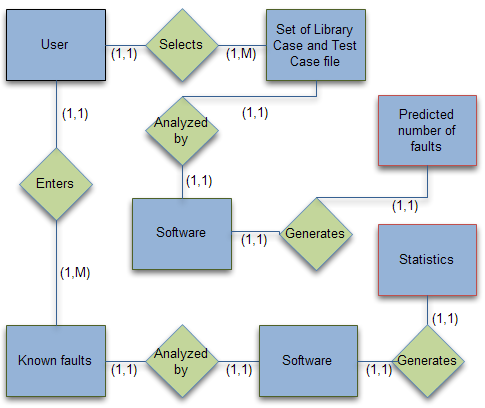
**3.1.1 Data Objects**

* Case library file - a collection of previously developed program modules stored in a flat text file using a space delimited and rows-columns format. All modules in the case library are represented by a given number of software complexity metrics and a software fault attribute. Using the row-column format, a given module, *i*, is composed of software metrics values, and a software faults value {x*i*1, x*i*2, x*i*3 … x*ik*, y*i*}, where *k* is the number of software metrics, x*ab* is the value in the ath row and bth column. The case library is used to predict number of faults for program modules with unknown faults and known metrics.
* Test case file - a collection of program modules for which the user wishes to obtain the predicted number of faults for each module, stored in a flat text file using a space delimited rows-columns format. All modules in the test case file are represented by a given number of software complexity metrics, which must be the same as the number of complexity metrics in the case library file modules. Using the row-column format, a given module, *i*, is composed of software metrics values {x*i*1, x*i*2, x*i*3 … x*ik*}, where *k* is the number of software metrics, x*ab* is the value in the ath row and bth column.
* Answer file - a collection of the same program modules as described by the test case file with the actual number of faults for each module. The answer file is stored in a flat text file with each module's number of faults on a separate line.

**3.1.2 Relationships**

1. The user selects a case library file and a test case file
2. The case library file and test case file are analyzed by software
3. The software generates the predicted number of faults
4. The User enters the answer file with number of known faults (if any)
5. The known faults are analyzed by software
6. The software generates statistics

**3.1.3 Complete Data Model**

****

***Figure 3.1.3-1: ERD Model***

**3.1.4 Data Dictionary**

* Case library file - a collection of previously developed program modules.
* Test case file - a collection of program modules for which the user wishes to obtain the predicted number of faults for each module.
* Answer file - a collection of the same modules as in the test case file, but with the known number of faults for each module.
* Program module - a component of a larger software application. A module may be a class, a function, or a source code file, for example.
* Software complexity metrics - a set of values {x­1, x2, ... xn­} that describe the complexity of a program module. Examples include source lines of code, required reliability, database size, and the number of different screens.
* Statistics – Consist of mean-squared error, standard deviation, and other basic statistics.
* Predicted number of faults – the number of faults that are predicted for each module in the test case file.

**4.0 Functional Model and Description**

There are three main software functions:

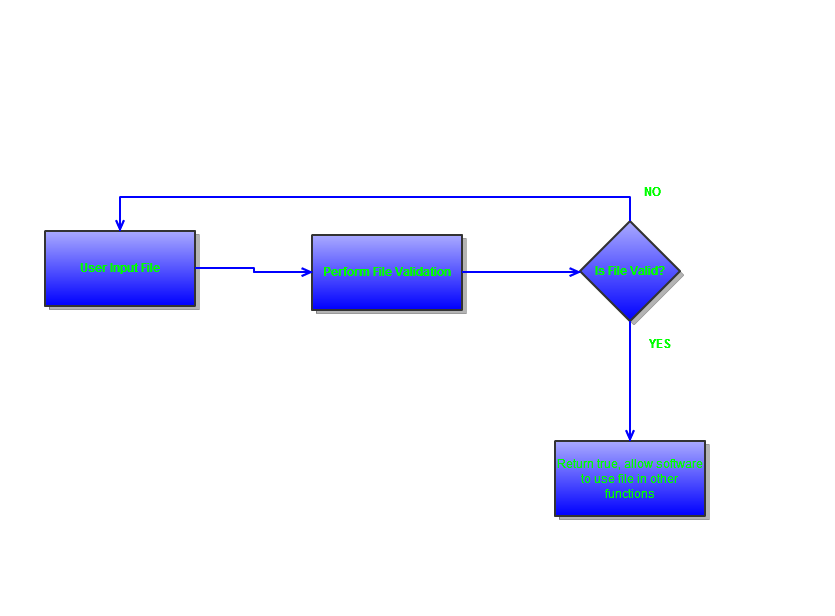
1. User-Input Validation – The initial input given by user (case library file, test data file, known number of faults, number of nearest neighbors used for prediction) is checked to ensure it is useable input. An example of invalid input would be a test data file that doesn’t exist.
2. Predict Number of Faults – The software will predict the number of faults for each program module based on the provided input from user.
3. Compute Statistics – The mean-squared error, standard deviation, and other basic statistics are calculated based on the predicted number of faults and the actual number of faults (provided by user).

**4.1 Description for Function: User-Input Validation**

**4.1.1 Processing narrative (PSPEC): User-Input Validation**

User input is tested to ensure it is valid input. Validation includes checking for file existence, readability, and proper format in both the case library file and test data file. Also validation occurs when user optionally selects and uploads the actual number of faults for the test data file modules. In the event of invalid input, the user is informed of error and again prompted for the respected input data.

**4.1.2 User-Input Validation flow diagram**



**4.1.3 User-Input Validation interface description**

All data is taken from user by file browser or manual input via text box. The function will return a Boolean variable (True or False). In the event of an error a message box will inform user of validation error and the user may enter data again. Data will be entered via a text box or file browser.

**4.1.4 User-Input Validation Performance Issues**

With large data files validation may take some time to properly analyze and validate the data.

**4.1.5 User-Input Validation Design Constraints**

The case library and test case file will need to be in the proper format to be properly analyzed by the software. This particular format must be documented well and easily accessible to anyone using the program.

**4.2 Description for Function: Predict Number of Faults**

**4.2.1 Processing narrative (PSPEC): Predict Number of Faults**

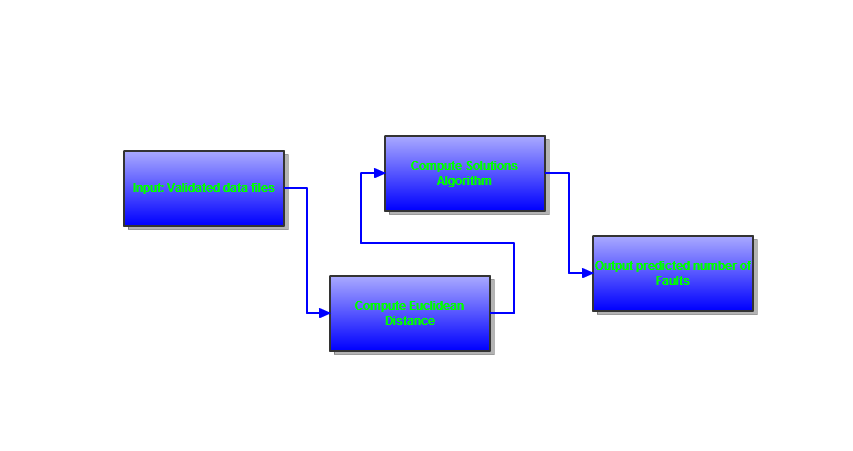
The software will use the validated files to predict the number of faults for each program module using the test case file. Euclidean distance is the similarity function that will determine the distance between a module and its nearest-neighbor. The solution algorithm will predict the number of faults in a program module based on a set of nearest neighbors.

Euclidean Distance Formula

Solution Algorithm

predicted number of faults formodule, nearest-neighbors

**4.2.2 Predict Number of Faults flow diagram**



**4.2.3 Predict Number of Faults interface description**

Input to this function is a valid case library file and test case file. Output from this file will be the predicted number of faults in each program module. This data will be displayed for the user to view. The user may also input the actual number of faults at this point. At this point the input will be validated and the statistics portion of the program will be called, otherwise this program or function will terminate.

**4.2.4 Predict Number of Faults Performance Issues**

If the user inputs a large file (i.e. case library, test file, or actual number of faults) the computation and analysis may take a considerable amount of time.

**4.2.5 Predict Number of Faults Design Constraints**

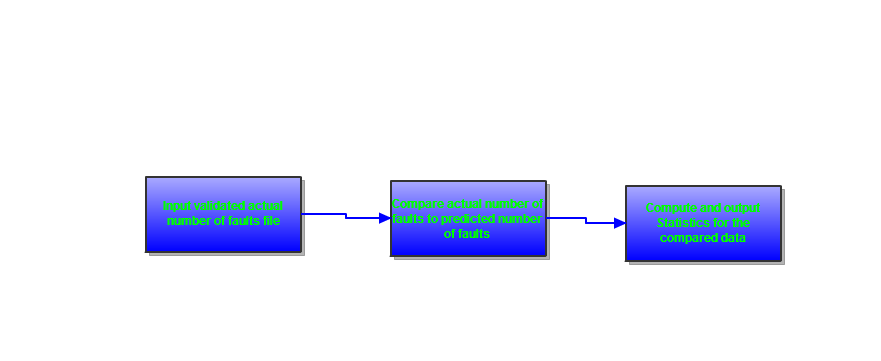
If given the actual number of faults the file must be formatted the same as the other inputted files. This file must also contain the same number of modules as the test module in order to be used correctly, not having the correct amount results in an error of invalid data.

**4.3 Description for Function: Compute Statistics**

**4.3.1 Processing narrative (PSPEC): Compute Statistics**

The predicted number of faults and actual number of faults from function 2 are used to calculate the mean-squared error and standard deviation of the data set. This function can only be called if the actual number of faults is given by the user.

**4.3.2 Compute Statistics flow diagram**



**4.3.3 Compute Statistics interface description**

Input to this function is the predicted number of faults via the test case file and the actual number of faults file. From these the mean-squared and standard deviation for the data set are calculated and outputted the user.

**4.3.4 Compute Statistics Performance Issues**

If the input data files are large, meaning they contain a large number of modules, it will take a long time to compute the statistics (mean-squared and standard deviation) for the data sets given.

**4.3.5 Compute Statistics Design Constraints**

The files must be valid so that the computation can be performed. This will be taken care of when function 1 is called for each inputted file.

**4.4 Software Interface Description**

**4.4.1 External machine interfaces**

External machines are not used since this is a single user program. All input and output is contained within the same machine and processed on the same machine. This could later be easily changed to process much larger data files across multiple machines using threads or dedicated machines for certain tasks.

**4.4.2 External system interfaces**

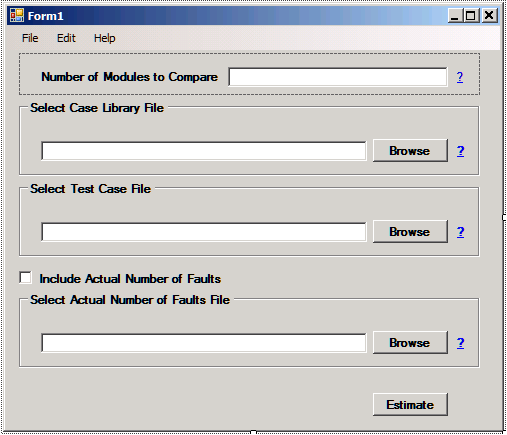
External machines are not used, so an external system interfaces is not required. In the future this could be expanded by copying the interface of this single-user application based on the needs of each external systems.

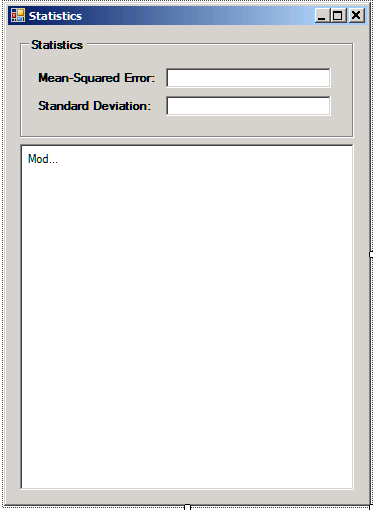
**4.4.3 Human interface**

The interface for this program will be graphical and allow for ease of use. The interface will contain an input box for all the functions necessary input. The actual number of faults is included at any point by checking a box to include it. The two main forms are shown below.

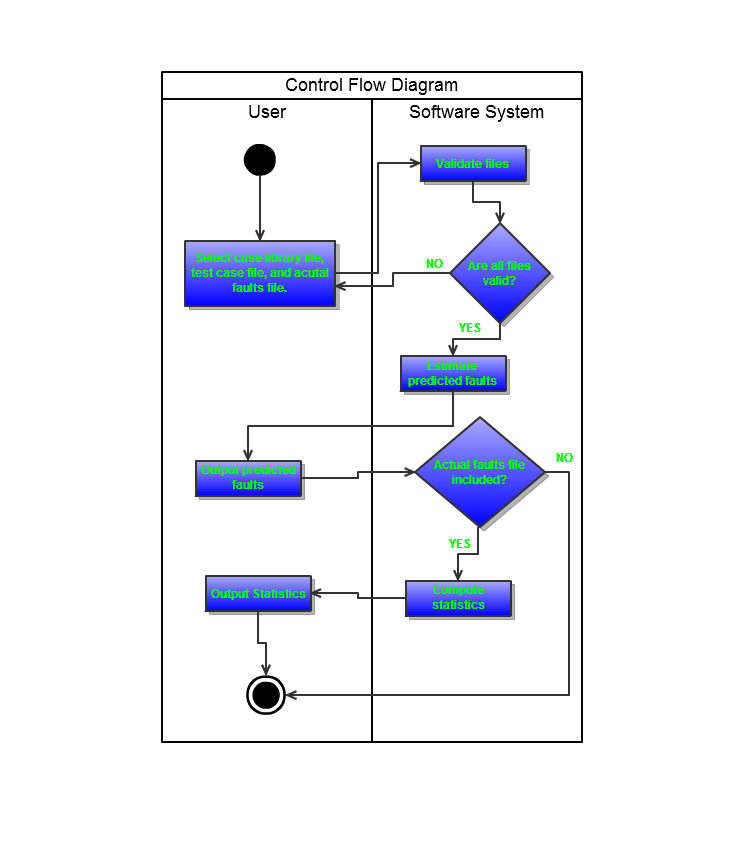
Form1 contains the number of Modules the user would like to compare, a place to enter a test case file and case library file. Again the checkbox will allow for the actual number of faults file to be added. Each browse button will bring up a file browser dialog box for easy adding of file paths. The estimate button will start the validations and calculations.

Form2 contains all output data. The computed statistics and break down of each module is given. The module break down will be viewable in a listbox with 5 different List-views for ease of use. This information can be easily saved by the user using the main form file menu options.

****



**4.5 Control flow description**



**5.0 Behavioral Model and Description**

**5.1 Description for Software Behavior**

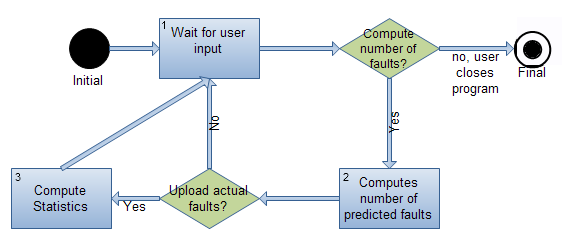
**5.1.1 Events**

* Application opens - controls are initialized and program waits for user input.
* The user clicks a button to select a case library file. The operating system has the Open File dialog box open and the user chooses a file. The user then clicks OK, where the program will check to see if the file is the correct format. The full file path name is presented so that the user can see which file was picked.
* The user clicks a button to select a test data file that contains the program modules that the predicted number of faults will be generated from. The operating system has the Open File dialog box open and the user chooses a file. The user then clicks OK, where the program verifies that the file is the correct format. The full file path name is presented so the user can see which file was picked.
* The user clicks a button to request that the software generate the predicted number of faults within the test data file. The software reads in the case library file, and test data file into memory, and calculates the predicted number of faults for each module. After processing all of the data, a table is presented to the user which shows the predicted number of faults for each module in the test data file.
* The user can now upload the actual number of faults of the test case file, should they be known. The operating systems File Open dialog opens and allows the user to select a file. After the user clicks OK, the program verifies that the file is of correct format. The full path name is presented so the user can see which file was picked.
* The user clicks a button to tell the software to compute statistics. The answer file containing the actual number of faults is used to calculate the mean-squared error, standard deviation, and other basic statistics for the predicted number of faults with the actual number of faults.

**5.1.2 States**

* **State 1:** Waiting for user input – occurs when the program opens and after a full iteration of the program.
* **State 2:** Computing predicted number of faults - occurs when the user supplies a case library file and a test case file and requests the program to predict the number of faults in the modules described in the test case file.
* **State 3:** Computing statistics - occurs after the program has predicted the number of faults and the user has supplied the answer file containing the actual number of faults for each module and requested the program to compute some basic statistics.

**5.2 State Transition Diagrams**

****

***Figure 5.2-1: State Transition Diagram***

**5.3 Control Specification (CSPEC)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Input Events** | **State 1** | **State 2** | **State 3** |
| User browses file system for case library file | **✓** |  |  |
| User browses file system for test case file | **✓** |  |  |
| User requests program to calculate predicted faults |  | **✓** |  |
| User browses file system for file containing actual number of faults | **✓** |  |  |
| User requests program to compute statistics |  |  | **✓** |
| **Output Events** | | | |
| Show file path | **✓** |  |  |
| Show predicted faults for each test case module |  | **✓** |  |
| Show statistics |  |  | **✓** |
| **Process Activation** | | | |
| Display open file dialog | **✓** |  |  |
| Compute predicted number of faults |  | **✓** |  |
| Compute statistics |  |  | **✓** |

***Figure 5.3-2: CSPEC Table***

# 

# 6.0 Restrictions, Limitations, and Constraints

**6.1 Time Limits**

We only have about two months to finish all documentation, bug fixings and enhancements. This creates severe limitations on the functionality that we will be able to implement.

**6.2 Budgetary Constraints**

We do not have a budget, and will therefore use open source and free software to implement the application.

**6.3 Hardware Constraints**

The program must be able to run on a typical desktop computer.

**6.4 Memory Constraints**

Limited size of RAM may result in performance stuttering or crashing when performing data analysis and computations. Also means that there is a bound on the number of data modules acceptable before freezing.

* Acceptable input is limited to forms following the “x1, x2… xn-1, xn, y” format. An additional column “Actual number of faults” are acceptable following the computation for predicted number of faults.
  + Unless otherwise noted by customer, we assume that they will use common format spreadsheet or text file as the test file.

# 7.0 Validation Criteria

Validation criteria are established during requirements analysis. What is the right product? The product must compute the predicted number of faults in a program module based on two given files. Assuming that these files (Library case and Test case) are in the accepted format “x1, x2, …, xn-1, xn, y”, the software proceeds to compute predictions based using the Similarity Function and Solution Algorithm. Mean-squared-error, standard deviation, and other basic statistics are calculated upon being provided the known number of faults by the user, through a file following the previously accepted format “x1, x2, …, xn-1, xn, y”.

**7.1 Classes of tests**

The types of tests to be conducted are specified, including as much detail as is possible at this stage. Emphasis here is on black- box testing.

* Unit Testing – The individual components to be tested are the read-in function (pick and choose file to upload to software), the file verification function, the Similarity Function, the Solution Algorithm, the known-faults acceptor, and the Statistics calculator function.
  + Develop drivers or stubs for these functions and use black-box testing:
  + Test Read-in function to ensure that it can correctly open and process files.
  + Test Verification function to determine that files fit the required format. Additionally check to ensure that number of modules to be compared does not exceed a certain value “X” that may cause instability. Warn user if this occurs.
    - Test Similarity Function with varying data that is within, at, and beyond the bounds of what is acceptable input.
    - Solution Algorithm tested in similar manner as Similarity Function (boundaries test).
    - Test read-in function (for known faults) to see that it works correctly. This test should be nearly identical to the first Read-in function.
    - Test output Statistics function by ensuring that the values are correct. Will test by boundaries.
* Integration Testing - Incrementally add more functions to test in a group.
  + Test each layer of the system decomposition individually before merging the layers
* Validation Testing – check to ensure functions conform to specifications by using a configuration review.
* System Testing – perform tests gauging how well the software works with either strained resources or when dealing with a lot of data (a boundary limit).

## 7.2 Expected software response

When correctly subjugated to the testing, we should find that we can catch up to 80% of the errors.

## 7.3 Performance bounds

* Limitations on number of modules to be processed cannot exceed an amount “X” (i.e. Large data file).
  + Exceeding this amount may cause instability, long process times, or cause the program to crash.
* The data’s values have a precision of up to 9 significant digits.
* Can accept positive or negative values as long as it conforms to fitting within 9 significant digits. Rounding errors may occur otherwise.