A Database for Managing Mutant Programs

Ronald Finkbine, Ph.D., Indiana University Southeast
rfinkbin@ius.edu

A mutant program is produced by modifying a correctly-functioning program by substituting an operator for another or exchanging a variable referenced within a statement. Mutation testing of a correctly-functioning program is performed by producing and a set of these mutant programs and examining their runtime behavior. The use of this type of testing can be used to determine the strength of a test suite (series of test cases).

The mutation program is executed against the test suite and there can be three logical results of this; (1) one of the test cases fails showing that the test suite is sufficient to detect the mutant program, (2) all test cases pass and this indicates a weakness in the test suite which needs to be repaired by the addition of a new test case, or (3) all test cases pass and a meaningless mutation program was generated.

This project contains component for support of mutation testing:

- Database for maintaining secure source code program
- Generator for producing mutants
- Compiler for generating class files for execution
- Execution manager for running class files and capturing outputs
- Database for maintaining test suite for source program
- Test suite manager for addition/deletion of test cases
- Results manager for results of test case runs

In the area of software development, the first level of program testing is the unit test, performed by the programmer by executing their program against the test suite (collection of test cases). The initial testing shows that a program passes its test suite but cannot assure the quality of the test suite. The mutation testing of a program attempts to measure the strength of a test suite, thus providing the programmer with feedback and an opportunity to improve a test suite.

As a real-world example, a generous faculty member can have an extremely simple examination question in an effort to ensure the students give the correct answer. Asking a graduate student in mathematics “What is two plus two?” is a question of the proper form, and it is an answerable question. However, it is not an inappropriate question to test a graduate student in mathematics. In order to improve the quality of software we must test and improve the quality of the assigned test suite.

A program that has successfully passed it test suite at the unit level is a candidate for mutation testing. This type of testing uses a source program as input, introducing modifications to generate mutant versions of the program. Each mutant is compiled and executed against the test suite and this can result in one of three results:

- One of the test cases fails, indicating the test suite is of sufficient strength
- All test cases pass, indicating a weak test suite
- All test cases pass, indicating a meaningless mutant has been generated

Please note that the last two choices detailed above are similar and it would be necessary for a programmer to interpret the results of a mutation’s execution to determine if a meaningless equivalent mutation has been created.

This paper describes the database tables necessary to support mutation generation and execution.

TOKEN TABLE DESCRIPTION

This system is not a traditional compiler, we are keeping the tokens of a Java program but are not doing the full lexical, syntactical and semantic analysis done by modern compilers. Though the Java language is being processed and lines/columns are insignificant, this information is retained to assist with pretty-printing output.

```
>>create table TokensOfJava(
>>      TokenNumber int not null auto_increment,
>>      LineStart tinyint,
>>      TokenText char(80)) ;
```

MUTANT TABLE DESCRIPTION

This table supports mutant generation. An algorithm will examine the token sequence for operators that are mutable (i.e. addition sign) and insert into this table its successor. All arithmetic operator substitution will go in the ASCII sequence of the characters (plus, minus, divide, …). Similarly treated will be the logical operators (and, or, not).

```
>>create table MutantOperator(
>>      MutantOperatorNumber int not null auto_increment,
>>      TokenNumber int,
>>      LineNumber int,
>>      MutantOperator char(3),
>>      CompileStatus int) ;
```

PROGRAM HISTORY TABLE DESCRIPTION

This table is used to support the runtime behavior of the mutant program, (i.e. compile status and test case results). Also note that part each record of this table is filled before the mutant execution by the programmer (control file name, input file name and output file name).

```
>>create table ProgramExecutionHistory(
>>      ProgramExecutionNumber int not null auto_increment,
>>      ProgramOrMutant char(7),
>>      TestCaseNumber int,
>>      RunTime int,
>>      TestCaseResult int,
>>      TestCaseInputFileName char(32),
>>      TestCaseOutputFileName char(32),
>>      TestCaseControlFileName char(32)) ;
```

A subsequent phase of this project will treat the variables declared within the program as a sequence similar to the treatment given to the arithmetic operators.

This presentation and project are in support of using mutant program generation and management to determine the effectiveness of software test suites.