CREAM - A SYSTEM FOR OBJECT-ORIENTED MUTATION OF C# PROGRAMS

Abstract

Creator of Mutants (CREAM) is a tool that introduces small faults called mutations into C# programs. It is useful to investigate the quality of a mutated program and effectiveness of a prepared test suite. The CREAM system generates several object-oriented types of faults, puts them into the code of the original program according to its parsed trees, and compiles the modified sources. Obtained assemblies can be tested measuring the number of revealed modifications. CREAM cooperates with NUnit for running of test suites against the set of mutants.

1. INTRODUCTION

Mutation technique is used to investigate the quality of software and available test suits, and for comparing the effectiveness of testing methods [1-5]. It consists in introducing small faults into a program and then testing its modified versions - so-called "mutants". Well designed test suite should "kill" all the mutants that are not equivalent - i.e. whose behavior can differ from the behavior of the original program.

Types of the introduced faults are defined by mutation operators. In structural languages many mutation operators were used [5]. They dealt mainly with modified usage of constants and variables, and with replacement of logical, relational and arithmetical operators, for example "\geq\" instead of ">\". Such "traditional" operators were also successfully applied in object-oriented languages.

However, object-oriented languages provide also new constructions, like class declarations and references, information hiding, inheritance, polymorphism, method overloading. These constructions were not considered by those simple mutation operators. Object-oriented mutation operators were proposed testing of OO features [3,6-10].

Object-oriented mutation testing was supported by tools only for Java programs [11,12]. Therefore the CREAtor of Mutants system (CREAM) was designed and implemented [13]. It was intended to generate mutants for C# programs using object-oriented mutations. Application of advanced mutation operators is much more complicated
than the traditional ones. Program changes introduced by many operators depend on the structural information of the inheritance hierarchy of a program. This information can be remote from the modified code area and distributed in different parts of the program. In the first version of the CREAM system we begun with such advanced operators and therefore used a parser-based approach for mutant generation.

Mutated programs should be able to compile properly, otherwise they are invalid mutants. The designers of mutation systems try to avoid generating such mutants. Object-oriented mutation operators require also precise and comprehensive specification. Without checking specific conditions on mutation application we could obtain many equivalent or even invalid mutants.

The CREAM system puts several object-oriented types of faults into the code of the original program according to its parsed trees, and compiles the modified sources. Obtained mutants can be tested measuring the number of revealed modifications. CREAM cooperates with NUnit for running of test suites against set of mutants.

2. MUTATION TOOLS

Mutation technique based on traditional mutation operators is supported by several tools devoted to different programming languages. The Mothra system was implemented for generation and testing mutants for Fortran programs [14]. The Fortran program is translated to its intermediate form. Changes reflecting modification of a source code are introduced in that form. Execution of mutants is realized by interpretation of the program. Another tool is the Proteum system [15] used for a mutation analysis of C programs. Other mutation tools are Jester and Nester [16]. They support simple mutation operators in Java and C# programs, accordingly. They introduce mutations looking for appropriate program constructions and substituting them according to given regular expressions. In this method equivalent mutants or even invalid ones can be created. It is also not sufficient for more complex, object-oriented mutation operators. Another approach to analysis of Java programs is proposed in [17]. The results returned by a method are changed simulating a faulty program behavior.

To our best knowledge the only mutation tools with object-oriented operators were implemented for Java programs. An announced NMutator tool [18] supporting object-oriented mutation of C# code has not appeared yet. There are also announced other plans to develop a mutation system for .NET compatible with NUnit [19]. The mostly known tool for Java programs is MuJava [12]. It can use a comprehensive set of traditional and object oriented (so-called class) mutation operators. MuJava generates mutants either by direct modification of intermediate code of Java Virtual Machine, or creating one modified program for many mutations (so-called meta-mutant). The tests used in MuJava system are similar but not compatible with unit tests widely used within JUnit environment. A successor of MuJava - the MuGamma system [20] - is based on monitoring the program states. The prototype of MuGamma currently implements a selective set of traditional mutants for Java. Class mutations are anticipated to be implemented in the future.

Another solution is the ExMan framework [21] that defines the overall architecture supporting mutation testing process. It needs an external tool for mutants generation. It was tested with a tool for C programs [22], and other custom mutation tools for C and Java using a source transformation language, TXL [23]. In all cases simple, traditional operators were considered.
Mutation testing is a very labor-consuming process. Therefore its tool support is of very high importance. Despite the existing tools, the mutation approach is still of limited use in the industrial practice. We need fully automated and efficient tools for generation of mutants and tests evaluation. The suitable integration with tools aimed on different testing approaches (unit testing, code coverage estimation) should enhance the testing process. Apart from the performance issues the identification of equivalent mutants causes big problems. In general case it is not automated so far within the mutation tools.

3. MUTATION TESTING WITH CREAM SYSTEM

3.1 Object-oriented mutation operators

There are many object-oriented operators defined primarily for Java programs [3,6,7]. The most of them could have been be adapted for C# programs, also other advanced operators concerning the C# features were proposed [9,10]. They were precisely specified by pre- and post-conditions of a program transformation. Using such definitions prevents for generating invalid mutants (e.g. mutants that are killed by the compiler), which is not a trivial task for the complex mutation operators. Characteristics of the operators for C# programs we have studied analytically and in experiments [8].

The CREAM system intended to provide an engine for generation of object-oriented mutations for C# programs. Basing on the gathered experiences we selected and implemented several operators. The firstly chosen operators require access to information about the structure of the program. Therefore they were introduced using a parser-based approach. In the first version of the CREAM system the following operators were implemented:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOP</td>
<td>Overridden method calling position change</td>
</tr>
<tr>
<td>IPC</td>
<td>Explicit call of a parent’s constructor deletion</td>
</tr>
<tr>
<td>EOC</td>
<td>Reference comparison and content comparison replacement</td>
</tr>
<tr>
<td>JID</td>
<td>Member variable initialization deletion</td>
</tr>
<tr>
<td>IHD</td>
<td>Hiding variable deletion</td>
</tr>
</tbody>
</table>

As an example we discuss the IPC operator (Fig. 1). A constructor of the base class is called at the beginning of creating an object of its inherited class. The constructor is called explicitly or a default constructor is used. The IPC operator deletes a direct calling of the constructor having at least one parameter. In C# the initialization list is used for constructor...
calling. Deleting a direct calling of a constructor with some parameters will cause a compilation error, if the constructor without parameters is not explicitly defined in the base class. Therefore, the IPC operator is applicable if the base class has at least two overloaded constructors, including one without parameters. The operator should not delete a constructor call without arguments, because it could generate mostly equivalent mutants.

In the CREAM system a parser creates syntax trees from the original source code (Fig. 2b). For a program there can be many compilation units and many syntax trees. All syntax trees of the program should be accessible in order to apply the object-oriented mutation operators.

A set of syntax trees is not convenient for inspecting of class hierarchy. Therefore using the parsed information the inheritance hierarchy of the program classes is created (Fig. 2a). It can be used for looking for a base class or an inherited class of a given class. Such class relations are searched while verifying the conditions of operators' application.

![Class Hierarchy](image)

![Parse tree](image)

Fig.2.a) A class hierarchy of a mutated program  
Fig.2.b) A parse tree of a mutated program

The syntax trees uniquely identify the program structure. Therefore the preconditions of the object-oriented operators can be verified and the correct syntax preserved in the mutated code. The applied mutation operators modify appropriate branches in a tree. Further the parser restores the source code of the mutant from the modified tree.

### 3.2 Mutation process overview

Mutation process using the CREAM system constitutes a sequence of steps. Selected steps are only due to observation and checking purposes and can be omitted in a target, automatic process.

**Process configuration** The mutation process is controlled by several options. We can set the options in order to limit the number of generated mutants. The option *Mutation limit for files* denotes the maximum number of mutants generated from each file for any mutation operator. *Mutation limit for whole programs* relates to the general number of mutants generated for a program. Using the *Compilation* option a user can determine whether the mutants created for a program should be compiled during the mutant
CREAM - a system for object-oriented mutation of C# programs

5
generation step. The general settings can be customized for the selected programs in the further steps, if necessary.

Program selection CREAM can mutate a single file programs with C# code (*.cs files) and the whole Visual Studio 2005 solutions. The chosen programs are loaded and shown on the list. We can mutate all or the selected, marked files of a program.

Operators selection We can use all mutation operators available in the tool or perform a selective mutation. In the later case we mark only selected operators to be applied on the programs. To assist comprehension of operators a full name emerges for a pointed operator (Fig. 3). All operators are also accompanied by examples illustrating their usage, similar to that from Fig. 1.

Process customization Whether some programs should have different options than the others the mutation process can be customized for them. We can select the files to be mutated, check Compilation option and set limit of mutants generated for each file and for a whole program if necessary.

Programs presentation Introduction of the object-oriented mutation operators depends on the program structure (Sec. 3.1). For a selected, loaded program we can view the parse tree for each properly parsed file of the program (Fig. 2b). On demand, the class hierarchy of the program can be also presented (Fig. 2a).

Mutants generation After setting up the mutation configuration the mutant generation can be launched. This process step can lasts for a long time and can be aborted if desired. The code of mutants will be stored in the required storage. The mutated programs for those the Compilation option was set are automatically passed to the compiler. Compiled solutions are stored accordingly.
Mutants verification

We can observe the resulting changes in the source code of a mutated program. The source code of the original program can be displayed in parallel to the mutated code. The changed code areas are marked with a different color allowing the code comparison (Fig. 4).

Fig. 4. Comparison of the source code of an original program and its mutant.

Testing the mutants with NUnit

The created mutants can be further used in any testing process. The CREAM system assists testing with NUnit tool. Test configuration can be set up or load from a file. It is used for specifying the details needed to run tests automatically. The appropriate mutated programs are passed to the NUnit tool called as a batch program. The execution results are compared with the pointed out results of the original program. The statistics of mutants executions are gathered and stored in the appropriate form for the further analysis.

4. CREAM ARCHITECTURE

The CREAM system consists of several modules (Fig. 5). The Parser module creates syntax trees for all program files and rebuilds the output code from the modified trees. In the first version of the CREAM a parser kcsparse\(^1\) was used. Created syntax trees are

\(^1\) ANTLR, kcsparse - an ECMA-334 C# Grammar Sample for ANTLR v2.7.6: www.antlr.org
analyzed by another module. It creates the inheritance class hierarchy of the mutated program and supports search operations in the parsed trees. These operations are used during realization of mutation operators. The MutationOperators module encapsulates the whole logic about the definitions of mutation operators. It can be easily extended with further object-oriented operators or with traditional ones.

The ProgramManagement module handles programs to be mutated. It can be a single source file with C# v1.1 code, or the whole solution VS 2005. A mutated program is passed to the C# compiler if required. The module manages also storing the mutated versions of the program. In the first CREAM version the programs are stored in the file system. They can be stored in a more effective way in an external storage tool. The module is responsible for providing appropriate interfaces for the cooperation with these external tools.

The whole mutation process is supervised by the ProcessControl module. It keeps the configuration information, like mutation limits, selected operators, lists of mutated objects, etc. It cooperates with Presentation module that comprises GUIs of the system.

The mutated programs can be further tested. The ProgramTesting module is responsible for the cooperation with the external testing tools. The first version of CREAM supports testing with the NUnit testing tool. The ProgramTesting module provides the appropriate interface, compares the results with the oracle of the original program and maintains statistics of the killed mutants.

The CREAM system was implemented in C# language. It requires for the execution .NET Framework 2.0. There are also used Visual Studio 2005 for compilation of mutants and NUnit system if tests for NUnit are executed.

5. CREAM VERIFICATION AND EVOLUTION

The CREAM system was experimentally verified on several projects, mainly open-source designs distributed with unit tests suites. The system was also used in the course on Dependability of Computer Systems. The course was taken by graduated students in fall 2006 in the Institute of Computer Science Warsaw University of Technology.

In all experiments no mutants were generated for IHD operator. It showed that the construction tested by this operator, hiding a field with the keyword new, was not used. For other mutation operators various numbers of mutants were created in dependence on the programming style and object-oriented mechanisms used in the programs. The ability of the published tests suites to kill the mutants varied from 90 to 10% for different programs. The main performance limitations concern the time and space for the storage of created mutants. Based on the gathered experiences the next generation of CREAM is under development. The main subjects of the system evaluation comprise:
• Usage of a new parser (NRefactory library from SharpDevelop) which satisfies the system requirements and supports the current version of C# language (2.0).
• Extension of the list of available operators; primarily with other object-oriented and advanced operators, and - for completeness - also with the traditional operators. Selected operators can be introduced using more efficient techniques than parser-based (e.g. reflection or meta-programming).
• Improvement of performance in mutated projects management. The projects can be stored in the developed repository based on Control Version System (Subversion) and therefore reducing requirements on the disc space.
• Optional introductory correlation with code coverage results. The mutation operators could be applied only to code areas that were covered in the previously performed test runs (before mutant generation).
• Correlation of mutated code areas with the results of code coverage for the given test suite.
• Processing the mutation results in comparison to other automated test results.

6. FINAL REMARKS

Application of mutation technique to the program development and maintenance can be not achieved without an automatic tool support. The CREAM system is the first attempt to use structural object-oriented mutations for C# programs. Using this system, the mutation has been applied to several real projects and their tests. The experiments resulted in some conclusions concerning the quality of the software and shown weak points of the attached test suites. The described tool is the base that evaluates to the advanced system of object-oriented mutation for C# programs.

BIBLIOGRAPHY


2 http://wiki.sharpdeverlop.net/default.aspx/SharpDevelop.NRefactory
3 http://subversion.tigris.org
**CREAM - AUTOMATYCZNY SYSTEM DO OBIĘKTOWYCH MUTACJI PROGRAMÓW W C#**

**Streszczenie**

CREAM (Creator of Mutants) jest narzędziem pozwalającym wprowadzać błędy tzw. mutacje do programów napisanych w języku C#. Może służyć do badania jakości mutowanego programu i efektywności przygotowanego zbioru testów. System generuje kilka obiektowych typów mutacji, modyfikując kod źródłowy zgodnie z drzewami rozbiór składniowego, a następnie kompiluje zmutowane programy. W procesie testowania można mierzyć liczbę błędów wykrytych przez testy. CREAM wspiera testowanie za pomocą środowiska NUnit.