Proposal of Integrated Software Testing Support Environment Based on the XML-based Source Code Representation

Ryota SASAKI†, Akihisa SUEHIRO†, Hirohide HAGA†, Shigeo KANEDA†

(Received May 27, 2010)

We propose the Integrated Software Testing Environment (IST) based on the XML-based source code representation of syntax information. The central part of IST is XML tagged source code database. In this paper, we describe two developments. One is a development of the conversion program, which converts original source code into mBML (miniBasic Markup Language), and the other is a mutant code generation program. In this generator, several mutant operators for mutation testing are implemented.

Key words : Integrated Software Testing Support Environment, XML, mBML, mutation test, mutant operator

キーワード：統合ソフトウェアテスティング支援環境，XML，mBML，ミューテーションテスト，ミュータントオペレータ

XML によるソースコード表現に基づく 統合ソフトウェアテスティング支援環境の構築

佐々木　亮太，　末広　暁久，　芳賀　博英，　金田　重郎

1　はじめに

近年の IT 社会において，ソフトウェアの品質が保証されていることが厳しく求められている．そのため，品質管理は企業にとって至上命題である．そこで，品質が要求基準を満たしていることを確認するために，一般的にソフトウェアテストという行程が用いられている．ソフトウェアはますます大規模かつ複雑化しており，典型的なプログラム開発プロジェクトにおいては，全費用の 50%以上の費用と全開発期間のほぼ 50%にあたる期間が，ソフトウェアのテストに費やされている [1)]．中には人命にかかわる重要なソフトウェアも存在し，致命的なバグによる事故が取り返しのつかない結果をもたらすこともある．2006 年には，シンドラーエレベータ株式会社のエレベータで，高校生が圧死して騒がれた事件が発生した．原因は制御盤のプログラムミスであった．この事件はマスメディアにも大きく取り上げられ，社会的にも大きな問題となった．その結果，シンドラーエレベータ株式会社は損害賠償の支払いはもちろん，企業ブランド価値の低下により，

† GraduateSchool of Engineering, Doshisha University, Kyoto
Telephone/FAX: 0774-65-6978, E-mail: rsasaki@ishss10.doshisha.ac.jp

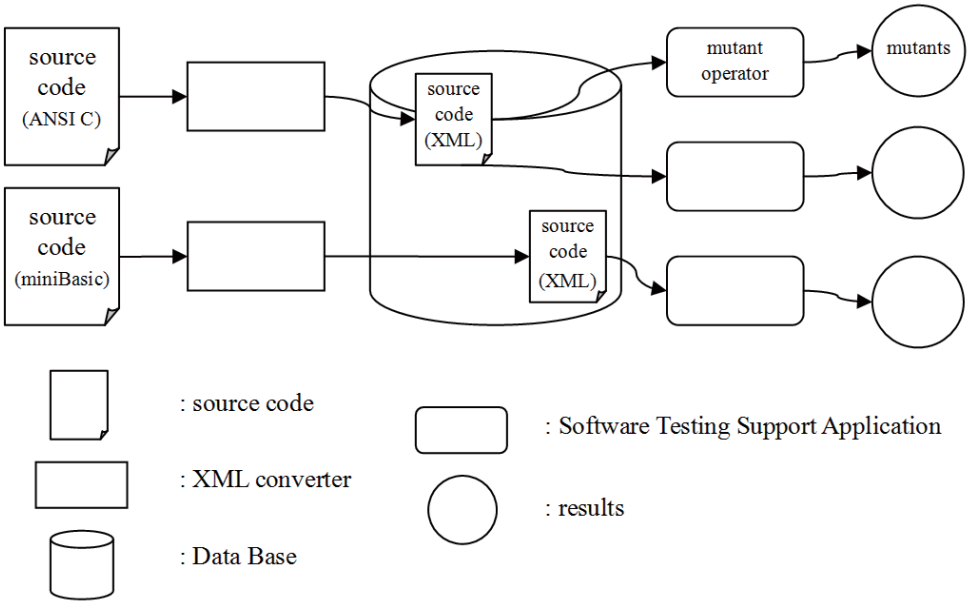

Fig. 1. Conceptual structure of IST

将来の商機を失う結果となった．また，被害者側の悲しみも甚大なものであり，再発防止が強く叫ばれている．この事件の原因であるプログラムミスはソフトウェアテストが不十分であったため発見できなかった．多大な費用と期間を要してソフトウェアテストを行っていても,このような致命的な事故が起こり得る．しかし，開発費用と期間は有限であるため．ソフトウェアテストを十分に行い，品質を保証できている企業やプロジェクトは少ない．

以上のような理由から，ソフトウェアテストによるソフトウェア開発の効率化にともなう品質向上を実現する手法が望まれている．そこで本研究室では，上記の問題を解決する統合ソフトウェアテスティング支援環境 IST(Integrated Software Testing Support Environment)の開発を進めている．

IST を構築するための第一歩として，様々な言語のプログラムソースを，汎用性の高い形式に変換する必要がある．そこで，汎用性の高い形式のひとつとして XML(extensive Markup Language) [6]に着目し，様々な言語のプログラムソースを XML に変換するための手法を開発した．

プログラムソースを XML 化するための既存のツールとしては，Java ソースコードを XML 化する JavaML コンパイラ [7]や，TeX で書かれた数式を XML 化する Tex2MathML[8]などがある．しかし，これらの既存のツールでは，特定の言語しか XML に変換することができず，さらにオープンソースではないのでシステムを拡張することができなかった．そこで，本稿では，構文解析器生成器（パーサジェネレータ）の一つである SableCC[9]を用いて，様々な言語に対応することができ，オープンソースとして二次利用可能な，プログラムソース XML 化ツールの構築を目的とする．

2　統合ソフトウェアテスト支援環境 IST

本研究で開発を進めている．IST の概念構成図を Fig. 1 に示す．IST では，以下の①から③の手順でソフトウェアテストの効率化を図る．

① あるプログラミング言語で書かれたプログラムソースを XML 化する．
② XML 化されたプログラムソースを DB に保存する．
③ XML ソースコードを対象としたソフトウェアテスト支援アプリケーションに適用する．

XML を利用する理由として，XML にはコンピュータ言語ではなく人間の理解できる言語を用いることによる可読性，独自のタグを作成したり，他の人が作成したタグを使用できる拡張性，要素や属性タグの追加･削除により変更要求に対応する柔軟性があること，XML を対象としたアプリケーションが数多く開発されているため二次利用が容易に行えるなどの利点があるからである．そのためプログラムソースを XML 化することにより，プログラムの構造の理解及びソフトウェアテストを支援するアプリケーションに利用することによる，ソフトウェアテストの効率化が期待できる．また，データベースにプログラムソースを保存する理由は，ソフトウェア開発が大規模かつ複雑になってきていることから，効率よく目的のデータを取得できるようにするためである．

3　プレーンテキストソースコードの XML 化

プログラムソースを XML 化するにあたって，本稿では，導出木(derivation tree)に着目する．これは，プログラムソースの解析情報における出力形式の一つであり，プログラムソースの XML 化をする際に，プログラムソースの解析情報を無駄なくシンプルに表現することが可能である．以降，導出木，システムの概要を述べた後，本ツールの開発に利用したソフトウェア SableCC について述べる．

3.1　導出木

導出木は，生成過程における記号の導出関係を表現した木である．BNF 記法を用いた四則演算の定義を Fig. 2 に，この文法に基づいた数式 `1+2*3` の導出木を Fig. 3 に示す．

3.2　システム概要

プログラムソース XML 化システムの実行の流れは以下の①と②の手順である．Fig. 2 の文法に基づいた数式 `1+2*3` の XML 化の手順を Fig. 4 に示す．

① プログラムソースから，生成規則に基づき，導出木を生成する．

② 深さ優先探索により，その導出木のノードをたどるごとに，XML において親ノードに子ノードを追加していき，導出の木構造を XML で表現する．

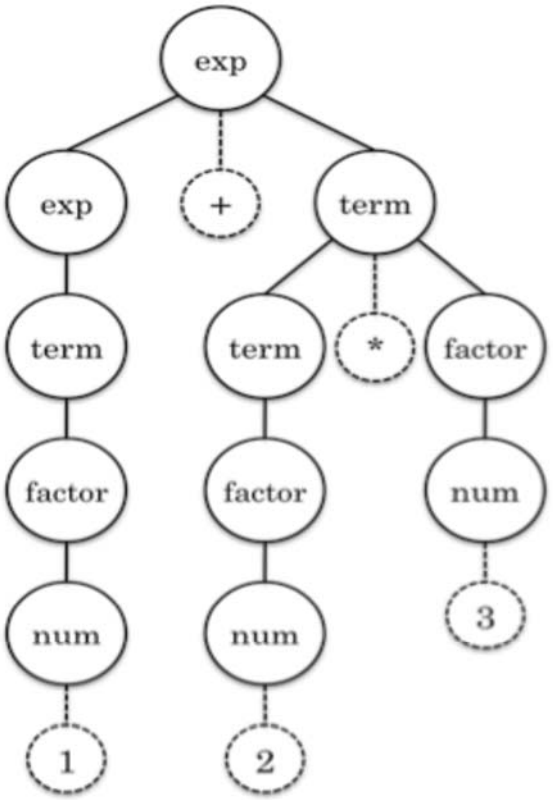

Fig. 2. Grammar of simple arithmetic expression

```
exp → exp + term |
      exp - term |
      term
term → term * factor |
       term / factor |
       factor
factor → num |
         (exp)
num → 0|1|2|3|4|5|6|7|8|9
```

Fig. 3. Derivation Tree of expression “1+2*3”

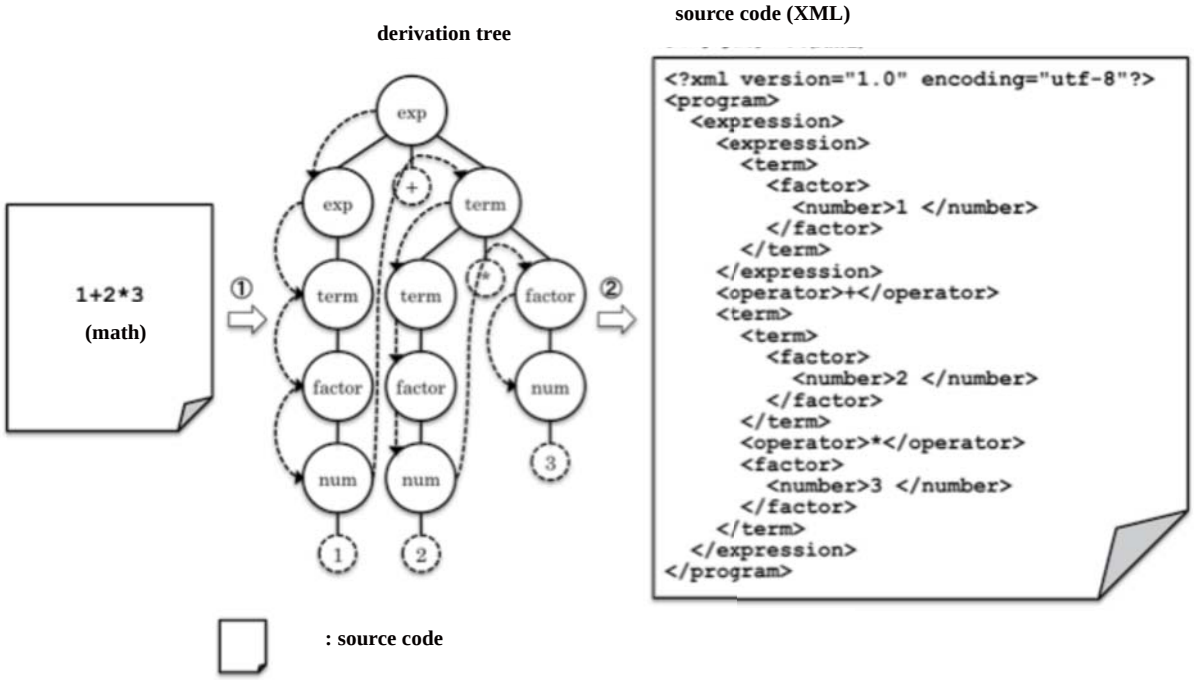

Fig. 4. Outline of XML Converter System

3.3 構文解析器生成器 SableCC の概要

SableCC は Java ベースの構文解析器生成器である．これは，BNF に基づいた文法ファイルを読み込み，その文法で書かれたデータを解析するための，Java または C#のプログラムソースを生成する．我々は，Microsoft.NET Framework 3.5 上で利用可能なクエリ言語である LINQ to XML を使用するため C#を使用し，利用するクラスを実装した．

SableCC で生成される主なクラスは，字句解析器 Lexer と，構文解析器 Parser，Parser によって生成される Node の導出木，それをたどるためのインタフェース Analysis である．

SableCC によるデータの解析手順は，まず，文法定義ファイルを読み込み，その文法で書かれたデータを解析するための Java または C#のプログラムソースを生成する．次に，それらの生成されたプログラムソースと，自分で作成した利用クラスを class ファイルにコンパイルする．最後に，実行時にそれらのオブジェクトに解析データを与え解析する．

4 SmBML の実装

miniBasic[10]のプログラムソースを XML で表現する文書構造化言語を mBML（miniBasic Markup Language）と呼ぶ．本稿では，提案した手法を用いて，miniBasic で書かれたプログラムソースを mBML に変換するコンバータプログラムである SmB2ML（Source Code of miniBasic to Markup Language）を実装した．以降，miniBasic を解説した後，mBML と SmB2ML の概要を述べる．

4.1 MiniBasic

miniBasic は，簡単な BASIC インタプリター（逐次翻訳・実行プログラム）である．以下に miniBasic の言語仕様を，Fig. 5 に miniBasic のプログラム例を示す．

データ型と演算子

- 定数は 0 を含む正整数とする
- 数値演算可能なデータ型は，整数型とする
- 数値演算子は，加減乗除（'+','-','*','/'）ならびに剰余('%')の 5 種類とする
- 数値演算子の優先順位は考慮しない

・数値演算式は，括弧（'(',')'）を用いることができる
・数値演算子より，括弧の評価を優先する
・数値演算の結果は，整数値を返す.
・論理演算子は，等号（'='）と不等号（'>','<'）の３種類とする
・論理演算の結果は，論理値（真／偽）を返す.
・文字列は，文字の列を引用符 "" で囲んだものとする

変数と代入
・変数の型宣言は不要とする
・代入前の変数の値はゼロをとるものとする
・変数への式評価値の代入は，代入記号 ':=' を用いて行う

予約語
・演算子と括弧 '+','-','*','/','%','=','>','<','(',')'
・代入記号 ':='
・条件実行文 'IF','THEN','ELSE','ENDIF'
・繰り返し実行文 'FOR','TO','NEXT'

組み込み関数名 'READ','PRINT','PRINTLN'

プログラム制御
・命令は，1 行に 1 命令とする.
・各命令の終わりは改行コードであるとする.
・1 行目から順番に実行する.
・プログラム終端（EOF）に達したら，プログラムは終了する
・条件実行文（IF）の書式は以下の通り
IF 条件 THEN [改行]
条件が真の場合に実行する命令列 [改行]
ELSE [改行]
条件が偽の場合に実行する命令列 [改行]
ENDIF [改行]
ただし，"ELSE [改行] 条件が偽の場合に実行する命令列 [改行]" の部分は空でもよい.
・繰り返し実行文（FOR）の書式は以下の通り（インクリメントのみ）
FOR 変数 := 開始値 TO 終了値 [改行]
命令列 [改行]
NEXT [改行]

組み込み関数
・関数名 READ 引数 変数
標準入力から整数値の読み込みを行う
・関数名 PRINT 引数 数値演算式
式の値を，標準出力へ表示する
・関数名 PRINT 引数 文字列
文字列の内容を，標準出力へ表示する
・関数名 PRINTLN 引数 なし
標準出力へ [改行] を表示する

```
PRINT "ISlab Testing group"
READ I
FOR X := 1 TO I
  FOR Y := 1 TO I
    PRINT X * Y
    PRINT " "
  NEXT
PRINTLN
NEXT
PRINTLN
```

Fig. 5. miniBasic

4.2 mBML

mBML は miniBasic のプログラムソースを XML で表現する文書構造化言語である．miniBasic のプログラムソースと mBML の要素の対応を Table 1 に示す.

4.3 SmBML の概要

SmB2ML は miniBasic のプログラムソースを mBML に変換するコンパイラである．これを 3 章の手段を用いて実装した．なお，SableCC による miniBasic のプログラムソースの解析のための文法ファイルは，SableCC プロジェクトが評価用サンプルプロジェクトとして配布されている minibasic-1.0.grammar を利用した．SmB2ML を用いて，Fig. 5 の miniBasic プログラムソースを mBML 化したプログラムソースを Fig. 6 に示す.

5 ソフトウェアテスト支援アプリケーション

IST において，XML で記述された C 言語のソースコードを対象としたソフトウェアテスト支援アプリケーションとして，ミューテーションテストにおけるミュータントを生成するミュータントオペレータを実装した．以降，ミューテーションテスト及び実装したアプリケーションのシステム概要について述べる．

5.1 ミューテーションテストの概要

ミューテーションテストとは，ソースコードへ故意に単純な変更を 1 箇所加えたプログラム（ミュータント）を新たに生成し，それを既存のテストセットが検出できるかどうかによって，テストセットを評価する手法である．従って，評価対象はソフトウェアではなく，テストセットである．ミューテーションテストはブランチカバレージなどの基準より品質の高いテストセットを構成できるため，高い品質が要求されるソフトウェアに適用するテストセット構成法である．

ミューテーションテストは，生成したミュータン

```
<program>
  <print>
    <string>"ISlab Testing group"</string>
  </print>
  <read id="I"/>
  <for>
    <from>
      <const val="1" type="integer"/>
    </from>
    <to>
      <var name="I" type="integer"/>
    </to>
    <for_statement>
      <for>
      ・・・
      </for>
      <println/>
    </for_statement>
  </for>
  <println/>
</program>
```

Fig. 6. mBML

Table 1. Specification of mBML

miniBasic プログラムでの役割	XML 要素名
識別子	var
数字	const
文字列	string
演算子	operator
標準入力	read
標準出力	print
代入文	assingment
条件実行文	if
繰り返し実行文	for

トすべてに対してコンパイルやテストケースの実行を行うので, テストセットの評価に多大な時間をを要する. また, 大量のミュータントが生成されるユニットテストレベルで行われる.

ミューテーションテストの結果をもとに, テストケースが少なければ追加し, 多ければ取り除く. テストケースが少ないときに追加する方法としては, ブランチカバレージなどの基準に沿って作ったテストセットにミューテーションテストを適用して, 検出できなかったミュータントを検出するテストケースを新たに加えて, テストセットのバグ検出能力をあげる方法がある. 逆にテスト実行時間を短縮させるためやテストケースの数を減らすためにも使われる. ミューテーションスコアを下げずにテストケースの数を最小にする手法ついてはミューテーション法を用いたテストセット構成支援に関する研究[11]などがある.

5.2 ミュータントオペレータ

ソースコードにどのような変更を加えるかを決定するものがミュータントオペレータである, ミュータントオペレータの規則に従って, 機械的にソースコードから変化を加える箇所を探し出し, 変化を加えてミュータントを生成する, 例えば, オリジナルのソースコードに a = 1 + b;というステートメントがあるとする. これを, 算術演算子を置換（+を-になど) するというミィータントオペレータを用いて生成したミュータントは, a = 1 – b;となる.

本稿で構築したシステムのミュータントオペレータを Table 2 に示す.

例としてミュータントオペレータ AOR (Arithmetic Operator Replacement)について, XML コードを用いたミュータント生成の方法を以下に示す.

（例：C 言語）

```
1 + a → 1 - a
```

（例：XML）

```
<const value="1" type="integer" />
<operater kind="+" />
<var name="b" type="integer" />
↓
<const value="1" type="integer" />
<operater kind="-" />
<var name="b" type="integer" />
```

元の C 言語ソースコード「1 + a 」構造情報を XML で記述したものが,

```
<const value="1" type="integer" />
<operater kind="+" />
<var name="b" type="integer" />
```

である. AOR は XML ツリーから operator 要素を探し出し, その kind 属性を「+, -, *, /, %」の中から自分自身(+)以外すべて選んで置換する. 従って, AOR は「1 + a」から「1 - a」「1 * a」「1 / a」「1 % a」の４つのミュータントを生成する.

5.3 ミュータント

生成されたミュータントでテストケースを実行し, オリジナルプログラムでのテストケースの実行結果と比較して, 一致しない結果が一度でも現れたら, そのミュータントは検出されたと見なす. すべてのテストケースを実行し, 一度も一致しない結果が現れなかった場合は検出できなかったと見なす.

しかし, どのようなテストケースでも検出できないミュータントが存在する. そのようなミュータントを等価ミュータントという. 等価ミュータントを機械的に取り除く事は困難なため, 本論文のシステムでは等価ミュータントは未検出ミュータントに分類する.

Table 2. Mutant Operators

識別子	内容
AOR	算術演算子(+, -, *, /)を置換する
ROR	関係演算子(<, <=, >, >=)を置換する
LOR	論理演算子(&&, ||)を置換する
ASR	代入演算子(=, +=, -=)を置換する
ABS	変数に，絶対値を求める関数 abs()を被せる
UOI	単項演算子(++, --)を付け加える
BCR	ループ内の break と continue を置換する
LCN	条件式や反復文の条件の真偽値を逆転させる
LNI	論理変数の真偽値を逆転させる
COI	キャスト演算子を付け加えて変数の型を変換する
AST	配列の添字を 1 加算あるいは 1 減算する
RSR	return 文を return 0 に置換する
ASD	代入分を 1 つ削除する
STD	if, for, while, do, switch 文を 1 つ削除する
VRR	変数名(reference)を他の変数名に置換する
CLR	定数(literal)をプログラム中の他の定数と置換する

5.3.1 ミューテーションスコア

ミューテーションテストの結果は, 以下の式で計算されるミューテーションスコア(MS)で示される.

$$MS = \frac{\text{検出ミュータント数}}{\text{総生成ミュータント数} - \text{等価ミュータント数}} \quad (1)$$

ミューテーションスコアは 0 以上 1 以下の実数で示され，0 は全てのミュータントが検出されなかった事を，1 は全てのミュータントが検出されたことを表す.

5.4 システム概要

実装したミューテーションツールへの入力は，C 言語のソースコードとテストセットである. 出力は, ミューテーションスコアと, 検出できなかったミュータントの詳細である. 以下の①から⑥の手順でミューテーションテストを実行する.

① オリジナルのプログラムでのテスト結果を記録する.
② C 言語ソースコードを XML へ変換する.
③ ミュータントを生成する.
④ XML コードを C 言語ソースコードへ変換する.
⑤ ミュータントでテストセットを実行し, 違いを検出できたミュータントを分別する.
⑥ ミューテーションスコアを算出する.

6 考察

6.1 XML への変換

本研究では，SableCC を用いて SmB2ML を実装した. 本章では SableCC と他の構文解析器生成器によるプログラムソース XML 化ツールの効率について比較し，SableCC による SmB2ML の有効性を検討する. 他の構文解析器生成器として，構文解析器生成器の一つである yacc を使用する. yacc は構文解析のみを行うので, 字句解析に字句解析を行うプログラムを自動生成する lex を使用する.

yacc/lex[12]によるデータの解析手順は，生成規則に基づいた導出木を生成し, それをボトムアップでたどり，各ノードに対応した処理を行う. プログラムソースを XML 化する上でこれは大きな問題となる. なぜなら，子どもから母が生まれることがないように，XML を扱う上でも，子ノードから親ノードの生成はできず, ノードの情報を保持できる配列に随時ノードのデータを格納していき, 探索を終えた時点で, 格納されたデータよりプログラムソースを XML 化するためである. 一方，SableCC では生成した導出木をトップダウンでたどり, 各ノードに対応した処理を行うので, この問題に左右されることはない. よって，SableCC は yacc/lex より効率的にプログラムソースの XML 化ツールを実装できる. また, SableCC はデータを解析するための Java または C#のプログラムソースを生成するため，LINQ to XML などを活用できる利点もある.

6.2 ミュータントオペレータ

C 言語で記述した 83 行の関数と 32 件のテストケースに, 本論文の提案する手法を実装したシステムを適用して実験を行った．評価基準は実行時間，保守性, 拡張性, 読解性（ソースコードの読みやすさ）である．

このテストセットは，100%ブランチカバレージになるように作ったテストセットに, 限界値分析を行ったテストセットを加えて作ったものである．

今回の実験では, XML 化したコードの行数は 835 行となり，ミュータント総数 1515 の内，検出済みミュータント群に 1475，未検出ミュータント群に 40 のミュータントが割り振られ，ミューテーションスコアは 0.974 となった．そして，ミューテーションテストの全工程の実行時間は 7 分 7 秒であった. そのうちコンパイルとリンクにかかった時間が大半である．生成したファイルの総容量は 112MB，使用したハードウェアは，Intel® Core™2 CPU 6600 2.40GHz, 1.00GB RAM である．

本システムでは, 等価ミュータントを未検出ミュータントとして扱ったが，未検出ミュータント 40 個を調べた結果，その内等価ミュータントは 33 個存在した．以下に例を 1 つ示す．

（オリジナル）
```
double geturei = 0.0;
if(year < 2000){
  geturei = (double)saday – 24.1;
}
```
（ミュータント）
```
double geturei = 0.0;
if(year < 2000){
  geturei += (double)saday – 24.1;
}
```

上記の例は, ゼロで初期化している変数に初めて代入する箇所で，代入演算子「=」を「+=」に置換したミュータントである．この場合，オリジナルとミュータントは常に同じ結果になるため, どのようなテストケースもこれら 2 つの違いは検出できない．よって，このミュータントは等価ミュータントとなる．ちなみに，観測対象のデータ型が実数の場合は, オリジナルの結果とミュータントの結果の差が ε 以下の場合に, 両者の結果に違いが見られなかったと判断する．ε は，ミューテーションテスト実施社が, テストセットのテスト対象プログラムによって，任意に決める．本実験では 0.1 とした．

今回生成された等価ミュータント 33 個のうち 29 個は，ミュータントオペレータ ABS によって生成されたものであった. このような過度に等価ミュータントを生成するミュータントオペレータには改善が必要である．

また，ミューテーションテストの後に，未検出ミュータントが検出するテストケースを新たにテストセットに加え, テストセットの品質を向上させる. これは自動ではなく，人の手によって行う．今回の実験では，7 件の未検出ミュータント（等価ミュータントは除く）が生成された. 以下に例を 1 つ示す.

（オリジナル）
```
if(year < 2000){
  saday -= day – 1;
}
```
（ミュータント）
```
if(abs(year) < 2000){
 saday == day – 1;
}
```

上記の例は，year の値が-2000 以下で，かつ day の値が 1 でないテストケースが, テストセットに存在しなかったため, 上記のミュータントを検出することができなかった．例えば，year = 3000，day = 5 となるテストケースを新たに加えれば, このミュータントを検出するようになる．

同様に, 他の 6 件の未検出ミュータントも検出するように, 合計 5 件のテストケースをテストセットに新たに加え, 再度ミューテーションテストを行った．その結果，未検出だったミュータント 7 件が検

出できるようになったため，ミューテーションスコアは0.978となり，テストセットの品質が向上した．

7 おわりに

本稿では，プログラムソースXML化ツール，及び，ソフトウェア支援アプリケーションのミューテーションテストにおけるミュータントを生成するミュータントオペレータを実装し，検討した．これにより，IST開発の第一歩を踏むことができた．今後は，様々な言語のプログラムソースXMLツールの実装はもちろんのこと，ISTの開発を進めていく．

参考文献

1) Glenford J. Myers 著，長尾真 監訳，松尾正信 訳，“ソフトウェア・テストの技法 第2版”，(近代科学社，東京，2006)，p.iii.
2) Boris Beizer 著，小野間彰/山浦恒央訳，“ソフトウェアテスト技法”，(日経 BP，東京，1994)，p.49-97
3) Aditya P. Mathur, “Foundations of Software Testing”, (PEARSON Education, NJ, 2008), p.502-593
4) Paunl Ammann, Jeff Offut, “Introduction to software testing”, (Cambridge University Press, UK, (2008), p170-212
5) 山中祐介，大畑文明，井上克郎，“プログラム解析情報のXMLデータベース化 –提案と実現–”，コンピュータソフトウェア，Vol.19，p39-43（2002）
6) “http://www.w3.org/TR/REC-xml/”
7) Greg J. Badros, “JavaML: A Markup Language for Java Source Code”, Computer NetWorks, Vol.33, No1-6, p159-177 (2000)
8) “http://www.cmt.phys.kyushu-u.ac.jp/~M.Sakurai/java/sdoc/mathml/test.html”
9) “http://sablecc.org”
10) “http://sablecc.sourceforge.net/grammars/minibasic.sablecc.web.html”
11) 斉藤孝志，大久保弘崇，粕谷英人，山本晋一郎“ミューテーション法を用いたテストセット構成支援に関する研究”，情報処理学会研究報告，p.1-8（2005）
12) “http://dinosaur.compilertools.net/”