Identifying Non-Essential Changes Using Semantic Code Clone Detection

Yungbum Jung
Programming Language Laboratory
Seoul National University
@ROSAEC Workshop
의미 코드 쌓 탐지 기술로 불필요한 코드 변화 찾기

정영범

서울대학교 프로그래밍 연구실
6회 ROSAEC 워크샵 @ 파주
의미 코드 쌍 탐지 기술로 불필요한 코드 변화 찾기

박사 정영범

서울대학교 프로그래밍 연구실
6회 ROSAEC 워크샵 @ 파주
MeCC: Memory Comparison-based Clone Detector

Haejung Kim, Yungbum Jung, Sungbum Kim, Kwangkun Yi
(Seoul National University, Seoul, Korea)
(sungbum@seoul.ac.kr, kwangkun@seoul.ac.kr, ykim@seoul.ac.kr)
The Hong Kong University of Science and Technology, Hong Kong
(kimh@hkusua.hku.hk)

ABSTRACT

In this paper, we propose a new semantic clone detection technique by comparing programs' abstract memory states, which are the memory states of the programs that are read and written during their execution. Our experimental study using three large-scale open-source projects shows that our technique can detect semantic clones that existing syntactic- or semantic-based clone detectors miss. Our technique is able to detect clones that are functionally similar but syntactically different. A few existing approaches to detect semantic clones [3], those based on program dependence graph (PDG) [23], or those based on program behavior graph (PBG) [24, 25], try to detect semantic clones. They are not based on abstract memory states. Our technique compares textual tokens from source code to determine code similarity. For example, Clonecatcher extracts and approximates textual tokens from source code to determine code similarity.

Categories and Subject Descriptors
C.4 [Software Engineering]: Software Engineering—versioning and change management; F.2 [Theory of Computation]: Theory of Computation—abstract data types and operations

General Terms
Languages, Algorithm, Experimentation

Keywords
Clones, memory, abstract interpretation, static analysis, software maintenance

1. INTRODUCTION

Detecting code clones is useful for software development and maintenance tasks such as identifying refactoring candidates [11], finding potential bugs [12, 14], and understanding software evolution [6].


Although these methods are good at detecting syntactically similar but functionally different code clones, they are not good at detecting semantic clones. Semantic clones are code clones that are functionally similar but syntactically different. A few existing approaches to detect semantic clones [3], those based on program dependence graph (PDG) [23], or those based on program behavior graph (PBG) [24, 25], try to detect semantic clones. They are not based on abstract memory states. Our technique compares textual tokens from source code to determine code similarity. For example, Clonecatcher extracts and approximates textual tokens from source code to determine code similarity.

Non-Essential Changes in Version Histories

Problem

Non-essential changes in version histories are often removed by a developer. Many tools or researchers may not notice or even consider the impact of these changes. Non-essential changes can affect the quality of change analysis, which is essential for software evolution studies.

Solution

We investigated the potential impact of non-essential differences in version histories. We found that up to 15% of changes in a large software project were non-essential. We implemented a tool to detect non-essential changes and evaluated its effectiveness. Our tool can detect non-essential changes in version histories.

Keywords
Non-essential modifications, software change analysis, impact of changes

1. INTRODUCTION

Many tools or researchers focus on essential changes, such as changes that involve code or files, or changes that are related to specific features or requirements. However, non-essential changes can also be important in software evolution studies.

Non-essential changes are changes that are not related to the software's functionality or its requirements. Non-essential changes can include changes that are made for maintenance purposes, changes that are made to improve code quality, or changes that are made for technical reasons.

Analyzing non-essential changes can provide valuable insights into software evolution. Non-essential changes can also be important for software maintenance tasks, such as code refactoring or code restructuring.

Analyzing non-essential changes can also be important for software evolution studies. Non-essential changes can affect the quality of change analysis, which is essential for software evolution studies.
소프트웨어 패키지에서 “지식”을 수확해왔다.
소프트웨어 패키지에서 “지식”을 수확해왔다.

코드가 썼었다?
[Eick et al. 2001]
SOFTWARE 패키지에서 “지식”을 수확해왔다.

함께 변경해야할 함수들 알려주기
[Zimmermann et al. 2004]
소프트웨어 패키지에서 “지식”을 수확해왔다.

버그 예측하기 등등...
소프트웨어 패키지에서 “지식”을 수확해왔다.
소프트웨어 패키지에서 “지식”을 수확해왔다.
Raw Material

Extraction

Resource
Resource → Extraction → Processing → Inference → Finished Product

\[ \{m_1, m_2\} \]

\[ m_3 \]
Finished Product

Raw Material

Resource

Processed Material

Inference

{m_1, m_2}

\[ m_3 \]

잘못된 정보를 원하지 않는 결과를 초래

\[ m_1 \rightarrow m_2 \rightarrow m_3 \]
Finished Product

Processed Material

Raw Material

Resource

Extraction

Inference

\{m_1, m_2\}

m_3

{

m_1

m_2

m_3

}

{

m

}

잘못된 정보는 원하지 않는 결과를 초래
Inference

 잘못된 정보는 원하지 않는 결과를 초래

Resource

Extraction

Raw Material

Processed

Finished Product

{m₁, m₂} m₃
Finished Product

Processed Material

Inference

Raw Material

Extraction

{m_1, m_2}

m_3
Finished Product

Processed Material

Inference

Raw Material

Extraction

{m_1, m_2}

m_3

잘못된 정보는 원하지 않는 결과를 초래
Finished Product

Processed Material

Inference

Raw Material

Extraction

\{
\begin{align*}
m_1, m_2 & \\
m_3 & 
\end{align*}
\}

잘못된 정보는 원하지 않는 결과를 초래
Finished Product

Resource

Raw Material

Extraction

Processed Material

Inference

\[ \{m_1, m_2\} \]

\[ m_3 \]

 잘못된 정보는 원하지 않는 결과를 초래
Processed Material

Filtered Material

Raw Material

Processing

Filtering

"Trivial" Diffs
변한 차이

**Version N**

Object field = ...

void sample() {
    List l = ...
    l.add(this.field);
    l.add(this.field);
    m(l.size());
    return;
}

**Version N+1**

void sample() {
    List l = ...
    l.add(this.field);
    m(l.size());
    return;
}

Object field = ...
번한 차이

Version N

```java
void sample() {
    List l = ...
    l.add(this.field);
    m(l.size());
    return;
}
```

Object field = ...

Version N+1

```java
void sample() {
    List l = ...
    l.add(this.field);
    m(l.size());
    return;
}
```

Object field = ...
변한 차이

Version N

void sample() {
    List l = ...
    l.add(this.field);
    m(l.size());
    return;
}

Object field = ...

Version N+1

void sample() {
    List l = ...
    l.add(this.field);
    m(l.size());
    return;
}

Object field = ...
Processed Material

Filtered Material

Raw Material

Processing

Filtering

Resource

“Trivial” Diffs
불필요한 변경

**Version N**

Object field = …

```java
void sample()
{
    List l = …
    l.add(this.field);
    m(l.size());
    return;
}
```

**Version N+1**

Object m_field = …

```java
void sample()
{
    java.util.List list = …
    list.add(m_field);
    int size = list.size();
    m(size);
}
```
불필요한 변경

Version N
Object `field` = ...

void sample()
{
    List l = ...
    l.add(this.field);
    m(l.size());
    return;
}

Version N+1
Object `m_field` = ...

void sample()
{
    java.util.List list = ...
    list.add(m_field);
    int size = list.size();
    m(size);
}
불필요한 변경

Version N
Object field = ...

void sample()
{
    List l = ...
    l.add(this.field);
    m(l.size());
    return;
}

이름 바꾸기 때문에 생기는 변경

Version N+1
Object m_field = ...

void sample()
{
    java.util.List list = ...
    list.add(m_field);
    int size = list.size();
    m(size);
}
불필요한 변경

**Version N**

```java
Object field = ...

void sample()
{
    List l = ...
    l.add(this.field);
    m(l.size());
    return;
}
```

이름 바꾸기 때문에 생기는 변경
간단한 키워드 변경

**Version N+1**

```java
Object m_field = ...

void sample()
{
    java.util.List list = ...
    list.add(m_field);
    int size = list.size();
    m(size);
}
```
불필요한 변경

Version N
Object field = ...

void sample() {
    List l = ...
    l.add(this.field);
    m(l.size());
    return;
}

이름 바꾸기 때문에 생기는 변경
간단한 키워드 변경

Version N+1
Object m_field = ...

void sample() {
    java.util.List list = ...
    list.add(m_field);
    int size = list.size();
    m(size);
}

간단한 타입 변경
불필요한 변경

**Version N**

```java
Object field = ...

void sample()
{
    List l = ...
    l.add(this.field);
    m(l.size());
    return;
}
```

이름 바꾸기 때문에 생기는 변경
간단한 키워드 변경

**Version N+1**

```java
Object m_field = ...

void sample()
{
    java.util.List list = ...
    list.add(m_field);
    int size = list.size();
    m(size);
}
```

간단한 타입 변경
지역 변수 이름 바꾸기
불필요한 변경

Version N
Object **field** = ...

void sample()
{
    List l = ...
    l.add(this.field);
    m(l.size());
    return;
}

이름 바꾸기 때문에 생기는 변경
간단한 키워드 변경
임시 변수 도입으로 생기는 변경

Version N+1
Object **m_field** = ...

void sample()
{
    java.util.List list = ...
    list.add(m_field);
    int size = list.size();
    m(size);
}

간단한 타입 변경
지역 변수 이름 바꾸기
DiffCat

Partial Program Analysis (PPA)

ChangeDistiller
DiffCat

Partial Program Analysis (PPA)

ChangeDistiller

V1

V2
DiffCat

Partial Program Analysis (PPA)

ChangeDistiller

V1 V2

소프트웨어 히스토리
DiffCat

Partial Program Analysis (PPA)

ChangeDistiller

소프트웨어 히스토리
DiffCat

Partial Program Analysis (PPA)

ChangeDistiller

소프트웨어 히스토리
**DiffCat**

- Partial Program Analysis (PPA)
- Refactoring Detection
- NED Detection
- ChangeDistiller

 сосфитвеар
히스тори
DiffCat

Files

Inferred Deltas
DiffCat

Files

1

v1

diff

v2

Inferred Deltas

1

1st Round Diffs
DiffCat

1. Files
   - v1
   - v2
   - rollback
   - diff

2. Inferred Deltas
   - Rename Refactorings
   - 1st Round Diffs
   - infer

3. v2
   - v2*
DiffCat

Files

v1

1

diff

4

diff

rollback

v2

v2*

Inferred Deltas

1st Round Diffs

2nd Round Diffs

Rename Refactorings

2

infer

1
Version N

Object field = ...

void sample() {
    field.foo();
    List l = ...
    l.add(this.field);
    m(l.size());
}

Version N+1

Object m_field = ...

void sample() {
    m_field.foo();
    List list = ...
    list.add(m_field);
    int size = list.size();
    m(size);
}
Version N

Object field = ...

void sample() {
    field.foo();
    List l = ...
    l.add(this.field);
    m(l.size());
}

Version N+1

Object m_field = ...

void sample() {
    m_field.foo();
    List list = ...
    list.add(m_field);
    int size = list.size();
    m(size);
}
Version N

Object field = ...

void sample() {
    field.foo();
    List l = ...
    l.add(this.field);
    m(l.size());
}

Version N+1

Object m_field = ...

void sample() {
    m_field.foo();
    List list = ...
    list.add(m_field);
    int size = list.size();
    m(size);
}
Version N

Object field = ...

void sample() {
    field.foo();
    List l = ...
    l.add(this.field);
    m(l.size());
}

Version N+1

Object m_field = ...

void sample() {
    m_field.foo();
    List list = ...
    list.add(m_field);
    int size = list.size();
    m(size);
}
**Version N**

Object field = ...

void sample() {
    field.foo();
    List l = ...
    l.add(this.field);
    m(l.size());
}

**Version N+1**

Object **field** = ...

void sample() {
    **field**.foo();
    List l = ...
    l.add(**field**);
    int size = l.size();
    m(size);
}
Version N

Object field = ...

void sample() {
  field.foo();
  List l = ...
  l.add(this.field);
  m(l.size());
}

Version N+1*

Object field = ...

void sample() {
  field.foo();
  List l = ...
  l.add(field);
  int size = l.size();
  m(size);
}
예제

**Version N**

Object field = …

```java
void sample() {
    field.foo();
    List l = …
    l.add(this.field);
    m(l.size());
}
```

**Version N+1***

Object field = …

```java
void sample() {
    field.foo();
    List l = …
    l.add(field);
    int size = l.size();
    m(size);
}
```
Version N

Object field = ...

void sample() {
    field.foo( );
    List l = ...
    l.add(this.field);
    m(l.size());
}

Version N+1*

Object field = ...

void sample() {
    field.foo( );
    List l = ...
    l.add(field);
    int size = l.size();
    m(size);
}

변경사항: 필요없는 this 없애기
```java
Version N

Object field = ...

void sample() {
    field.foo();
    List l = ...
    l.add(this.field);
    m(l.size());
}

Version N+1*

Object field = ...

void sample() {
    field.foo();
    List l = ...
    l.add(field);
    int size = l.size();
    m(size);
}
```

변경찾기   필요없는 this 없애기   임시 변수 사용
예제

Version N

Object field = …

void sample() {
    field.foo();
    List l = …
    l.add(this.field);
    - m(l.size());
}

Version N+1*

Object field = …

void sample() {
    field.foo();
    List l = …
    l.add(field);
    int size = l.size();
    m(size);
}
Version N

Object field = …

void sample() {
  field.foo();
  List l = …
  l.add(this.field);
  m(l.size());
}

Version N+1

Object field = …

void sample() {
  field.foo();
  List l = …
  l.add(field);
  int size = l.size();
  m(size);
}
## 불필요한 메소드 변경이 얼마나 많나?

<table>
<thead>
<tr>
<th>시스템</th>
<th>전체 함수</th>
<th>불필요한 함수</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ant</td>
<td>17 792</td>
<td>2 759 (15.5%)</td>
</tr>
<tr>
<td>Azureus</td>
<td>8 731</td>
<td>229 (2.6%)</td>
</tr>
<tr>
<td>Hibernate</td>
<td>15 881</td>
<td>1 153 (7.3%)</td>
</tr>
<tr>
<td>JDT-Core</td>
<td>8 837</td>
<td>673 (7.6%)</td>
</tr>
<tr>
<td>JDT-Ul</td>
<td>9 681</td>
<td>426 (4.4%)</td>
</tr>
<tr>
<td>Spring Framework</td>
<td>11 047</td>
<td>1 715 (15.5%)</td>
</tr>
<tr>
<td>Xerces</td>
<td>8 409</td>
<td>256 (3.0%)</td>
</tr>
<tr>
<td>합계</td>
<td>80 378</td>
<td>7 211 (9.0%)</td>
</tr>
</tbody>
</table>
불필요한 메소드 변경이 얼마나 많나?

<table>
<thead>
<tr>
<th>시스템</th>
<th>전체 함수</th>
<th>불필요한 함수</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ant</td>
<td>17 792</td>
<td>2 759 (15.5%)</td>
</tr>
<tr>
<td>Azureus</td>
<td>8 731</td>
<td>229 (2.6%)</td>
</tr>
<tr>
<td>Hibernate</td>
<td>15 881</td>
<td>1 153 (7.3%)</td>
</tr>
<tr>
<td>JDT-Core</td>
<td>8 837</td>
<td>673 (7.6%)</td>
</tr>
<tr>
<td>JDT-UI</td>
<td>9 681</td>
<td>426 (4.4%)</td>
</tr>
<tr>
<td>Spring Framewrk</td>
<td>11 047</td>
<td>1 715 (15.5%)</td>
</tr>
<tr>
<td>Xerces</td>
<td>8 409</td>
<td>256 (3.0%)</td>
</tr>
<tr>
<td>합계</td>
<td>80 378</td>
<td>7 211 (9.0%)</td>
</tr>
</tbody>
</table>
불필요한 메소드 변경이 얼마나 많나?

<table>
<thead>
<tr>
<th>시스템</th>
<th>전체 함수</th>
<th>불필요한 함수</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ant</td>
<td>17 792</td>
<td>2 759 (15.5%)</td>
</tr>
<tr>
<td>Azureus</td>
<td>8 731</td>
<td>229 (2.6%)</td>
</tr>
<tr>
<td>Hibernate</td>
<td>15 881</td>
<td>1 153 (7.3%)</td>
</tr>
<tr>
<td>JDT-Core</td>
<td>8 837</td>
<td>673 (7.6%)</td>
</tr>
<tr>
<td>JDT-UI</td>
<td>9 681</td>
<td>426 (4.4%)</td>
</tr>
<tr>
<td>Spring Framework</td>
<td>11 047</td>
<td>1 715 (15.5%)</td>
</tr>
<tr>
<td>Xerces</td>
<td>8 409</td>
<td>256 (3.0%)</td>
</tr>
<tr>
<td><strong>합계</strong></td>
<td><strong>80 378</strong></td>
<td><strong>7 211 (9.0%)</strong></td>
</tr>
</tbody>
</table>
불필요한 변경 제거하면 얼마나 좋나?

<table>
<thead>
<tr>
<th>Setup</th>
<th>Tot Rec</th>
<th>True Rec</th>
<th>Prec</th>
<th>Top 3</th>
<th>Only Err</th>
</tr>
</thead>
<tbody>
<tr>
<td>원래대로</td>
<td>93 576</td>
<td>20 501</td>
<td>0.219</td>
<td>0.442</td>
<td>0.220</td>
</tr>
<tr>
<td>불필요한 변경 제거</td>
<td>81 162</td>
<td>19 631</td>
<td>0.242</td>
<td>0.475</td>
<td>0.183</td>
</tr>
<tr>
<td>Δ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\frac{0.242}{0.219} = 1.105
\]
불필요한 변경 제거하면 얼마나 좋나?

<table>
<thead>
<tr>
<th>Setup</th>
<th>Tot Rec</th>
<th>True Rec</th>
<th>Prec</th>
<th>Top 3</th>
<th>Only Err</th>
</tr>
</thead>
<tbody>
<tr>
<td>원래대로</td>
<td>93 576</td>
<td>20 501</td>
<td>0.219</td>
<td>0.442</td>
<td>0.220</td>
</tr>
<tr>
<td>불필요한 변경 제거</td>
<td>81 162</td>
<td>19 631</td>
<td>0.242</td>
<td>0.475</td>
<td>0.183</td>
</tr>
<tr>
<td>∆</td>
<td></td>
<td></td>
<td><strong>+10.5%</strong></td>
<td><strong>+7.5%</strong></td>
<td><strong>-20.2%</strong></td>
</tr>
</tbody>
</table>

\[
\frac{0.242}{0.219} = 1.105
\]
불필요한 변경 제거하면 얼마나 좋나?

<table>
<thead>
<tr>
<th>Setup</th>
<th>Tot Rec</th>
<th>True Rec</th>
<th>Prec</th>
<th>Top 3</th>
<th>Only Err</th>
</tr>
</thead>
<tbody>
<tr>
<td>원래대로</td>
<td>93 576</td>
<td>20 501</td>
<td>0.219</td>
<td>0.442</td>
<td>0.220</td>
</tr>
<tr>
<td>불필요한 변경 제거</td>
<td>81 162</td>
<td>19 631</td>
<td>0.242</td>
<td>0.475</td>
<td>0.183</td>
</tr>
<tr>
<td>Δ</td>
<td>-13.3%</td>
<td>-4.6%</td>
<td>+10.5%</td>
<td>+7.5%</td>
<td>-20.2%</td>
</tr>
</tbody>
</table>

\[
\frac{0.242}{0.219} = 1.105
\]
불필요한 변경 제거하면 얼마나 좋나?

<table>
<thead>
<tr>
<th>Setup</th>
<th>Tot Rec</th>
<th>True Rec</th>
<th>Prec</th>
<th>Top 3</th>
<th>Only Err</th>
</tr>
</thead>
<tbody>
<tr>
<td>원래대로</td>
<td>93 576</td>
<td>20 501</td>
<td>0.219</td>
<td>0.442</td>
<td>0.220</td>
</tr>
<tr>
<td>불필요한 변경 제거</td>
<td>81 162</td>
<td>19 631</td>
<td>0.242</td>
<td>0.475</td>
<td>0.183</td>
</tr>
<tr>
<td>Δ</td>
<td>-13.3%</td>
<td>-4.6%</td>
<td>+10.5%</td>
<td>+7.5%</td>
<td>-20.2%</td>
</tr>
</tbody>
</table>

\[
\frac{0.242}{0.219} = 1.105
\]
정적 분석 기술로 소프트웨어 공학의 문제를 풀어보자!
ABSTRACT

In this paper, we propose a new semantic clone detection technique for comparing programs’ abstract memory states, which are computed by a semantic-based static analyzer. Our empirical evaluation using three large-scale open source projects shows that our technique can detect semantic clones that existing syntactic- or semantic-based clone detectors miss. Most clone detectors (12, 19, 24, 28, 32) are based on textual similarity. For example, CVCFinder (18) extracts and compares textual fragments from source code to determine code clones. Distinct (11) compares characteristic vectors extracted from abstract syntax trees (ASTs), yet they are not effective at detecting semantic clones that are functionally similar but semantically different. A few existing approaches to detect semantic clones (e.g., based on program dependence graphs (PDGs) (2), 20, 25) or by observing program executions via random testing (34) face limitations. PDGs can be affected by syntactic changes, and random testing lacks coverage. Our experimental study using three large-scale open source projects shows that our technique can detect semantic clones effectively. Memory Comparison-based Clone Detector (MeCC) is an effective semantic-based static analyzer that estimates the abstract memory states to determine clones. Since the abstract memory states have a collection of the memory states (through approximate) along the execution paths within a procedure, our technique can effectively detect semantic clones, and our clone detection ability is independent of static similarity of clone candidates. We implemented our technique as a clone detection tool, MeCC: Memory Comparison-based Clone Detector (MeCC), by extending a semantic-based static analyzer, Deckard (14). The extension is to support path-sensitivity and record abstract memory states to determine clones. We implemented our technique as a clone detection tool, MeCC: Memory Comparison-based Clone Detector (MeCC), by extending a semantic-based static analyzer, Deckard (14). The extension is to support path-sensitivity and record abstract memory states to determine clones. We implemented our technique as a clone detection tool, MeCC: Memory Comparison-based Clone Detector (MeCC), by extending a semantic-based static analyzer, Deckard (14). The extension is to support path-sensitivity and record abstract memory states to determine clones. We implemented our technique as a clone detection tool, MeCC: Memory Comparison-based Clone Detector (MeCC), by extending a semantic-based static analyzer, Deckard (14). The extension is to support path-sensitivity and record abstract memory states to determine clones. We implemented our technique as a clone detection tool, MeCC: Memory Comparison-based Clone Detector (MeCC), by extending a semantic-based static analyzer, Deckard (14). The extension is to support path-sensitivity and record abstract memory states to determine clones.

PyObject *PyBool_FromLong (long ok) {
    if (ok) result = Py_True;
    else result = Py_False;
    Py_INCREF(result);
    return result;
}

static PyObject *get_pybool (int istrue) {
    if (istrue == Py_True) return Py_True;
    else return Py_False;
    Py_INCREF(result);
    return result;
}
... *set_access_name(cmd_parms *cmd, void *dummy, const char *arg)*
    void *sconf = cmd->server->module_config;
    core_server_config *conf =
        ap_get_module_config(sconf, &core_module);
    const char *err = ap_check_cmd_context(sconf, NOT_IN_DIR_LOC_FILE | NOT_IN_LIMIT);
    if (err != NULL) {
        return err;
    }
    conf->access_name = apr_pstrdup(cmd->pool, arg);
    return NULL;
}

... *set_protocol(cmd_parms *cmd, void *dummy, const char *arg)*
    const char *err = ap_check_cmd_context(cmd, NOT_IN_DIR_LOC_FILE | NOT_IN_LIMIT);
    core_server_config *conf =
        ap_get_module_config(cmd->server->module_config, &core_module);
    char *proto;
    if (err != NULL) {
        return err;
    }
    proto = apr_pstrdup(cmd->pool, arg);
    ap_str_tolower(proto);
    conf->protocol = proto;
    return NULL;
}
... *set_access_name(cmd_parms *cmd, void *dummy, const char *arg)*
    void *sconf = cmd->server->module_config;
    core_server_config *conf =
        ap_get_module_config(sconf, &core_module);
    const char *err = ap_check_cmd_context(sconf,NOT_IN_DIR_LOC_FILE | NOT_IN_LIMIT);
    if (err != NULL) {
        return err;
    }
    conf->access_name = apr_pstrdup(cmd->pool,arg);
    return NULL;
}

statement
reordering

... *set_protocol(cmd_parms *cmd, void *dummy, const char *arg)*
    const char *err = ap_check_cmd_context(cmd,NOT_IN_DIR_LOC_FILE | NOT_IN_LIMIT);
    core_server_config *conf =
        ap_get_module_config(cmd->server->module_config, &core_module);
    char *proto;
    if (err != NULL) {
        return err;
    }
    proto = apr_pstrdup(cmd->pool,arg);
    ap_str_tolower(proto);
    conf->protocol = proto;
    return NULL;
... *set_access_name(cmd_parms *cmd, void *dummy, const char *arg){
  void *sconf = cmd->server->module_config;
  core_server_config *conf =
      ap_get_module_config(sconf, &core_module);
  const char *err = ap_check_cmd_context(sconf, NOT_IN_DIR_LOC_FILE | NOT_IN_LIMIT);
  if (err != NULL) {
    return err;
  }
  conf->access_name = apr_pstrdup(cmd->pool,arg);
  return NULL;
}

... *set_protocol(cmd_parms *cmd, void *dummy, const char *arg){
  const char *err = ap_check_cmd_context(cmd, NOT_IN_DIR_LOC_FILE | NOT_IN_LIMIT);
  core_server_config *conf =
      ap_get_module_config(cmd->server->module_config, &core_module);
  char *proto;
  if (err != NULL) {
    return err;
  }
  proto = apr_pstrdup(cmd->pool,arg);
  ap_str_tolower(proto);
  conf->protocol = proto;
  return NULL;
}
... *set_access_name(cmd_parms *cmd, void *dummy, const char *arg){
  void *sconf = cmd->server->module_config;
  core_server_config *conf =
      ap_get_module_config(sconf, &core_module);
  const char *err = ap_check_cmd_context(sconf, NOT_IN_DIR_LOC_FILE | NOT_IN_LIMIT);
  if (err != NULL) {
    return err;
  }
  conf->access_name = apr_pstrdup(cmd->pool, arg);
  return NULL;
}

statement                     intermediate                    statement                    splitting
reordering                    variables                        splitting

... *set_protocol(cmd_parms *cmd, void *dummy, const char *arg){
  const char *err = ap_check_cmd_context(cmd, NOT_IN_DIR_LOC_FILE | NOT_IN_LIMIT);
  core_server_config *conf =
      ap_get_module_config(cmd->server->module_config, &core_module);
  char *proto;
  if (err != NULL) {
    return err;
  }
  proto = apr_pstrdup(cmd->pool, arg);
  ap_str_tolower(proto);
  conf->protocol = proto;
  return NULL;
}
set_access_name(cmd_parms *cmd, void *dummy, const char *arg) {
    void *sconf = cmd->server->module_config;
    core_server_config *conf =
        ap_get_module_config(sconf, &core_module);
    const char *err = ap_check_cmd_context(sconf, NOT_IN_DIR_LOC_FILE | NOT_IN_LIMIT);
    if (err != NULL) {
        return err;
    }
    conf->access_name = apr_pstrdup(cmd->pool, arg);
    return NULL;
}

set_protocol(cmd_parms *cmd, void *dummy, const char *arg) {
    const char *err = ap_check_cmd_context(cmd, NOT_IN_DIR_LOC_FILE | NOT_IN_LIMIT);
    core_server_config *conf =
        ap_get_module_config(cmd->server->module_config, &core_module);
    char *proto;
    if (err != NULL) {
        return err;
    }
    proto = apr_pstrdup(cmd->pool, arg);
    ap_str_tolower(proto);
    conf->protocol = proto;
    return NULL;
}
불필요한 변경 예제

Version N

Object field = ...

void sample() {
    field.foo();
    List l = ...
    l.add(this.field);
    m(l.size());
}

Version N+1

Object m_field = ...

void sample() {
    m_field.foo();
    List list = ...
    list.add(m_field);
    int size = list.size();
    m(size);
}
불필요한 변경 예제

Version N

Object field = ...

void sample() {
    field.foo();
    List l = ...
    l.add(this.field);
    m(l.size());
}

Version N+1

Object m_field = ...

void sample() {
    m_field.foo();
    List list = ...
    list.add(m_field);
    int size = list.size();
    m(size);
}

보기에는 다를 수 있는데 하는 일은 같은 함수
불필요한 변경 예제

Version N

Object field = ...

void sample() {
    field.foo();
    List l = ...
    l.add(this.field);
    m(l.size());
}

Version N+1

Object m_field = ...

void sample() {
    m_field.foo();
    List list = ...
    list.add(m_field);
    int size = list.size();
    m(size);
}

보기에는 다를 수 있는데 하는 일은 같은 함수

의미 코드 쌬 (semantic code clone)
유사 코드 쌓 찾는 방법

프로그램

유사 코드 쌓

함수들

P1 P2
P3 P4

위치

정적 분석기

요약 메모리

\( \mathcal{F}(\vec{P}) = \vec{M} \)

메모리 비교

비슷한 것끼리 묶기

\( S(\vec{M}, \vec{M}') \)

유사도
실험 대상

<table>
<thead>
<tr>
<th>Projects</th>
<th>KLOC</th>
<th>Procedures</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>435</td>
<td>7,657</td>
<td>interpreter</td>
</tr>
<tr>
<td>Apache</td>
<td>343</td>
<td>9,483</td>
<td>web server</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>937</td>
<td>10,469</td>
<td>database</td>
</tr>
</tbody>
</table>
저는 유사 코드 쌍

Total 623 code clones

39%

53%

6% 2%

Type-1 Type-2 Type-3 Type-4

다른 도구들과 비교

CCfinder textual tokens

PDG-based program dependency graphs

DECKARD characteristic vectors

<table>
<thead>
<tr>
<th></th>
<th>CCfinder</th>
<th>PDG-based</th>
<th>DECKARD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>CCfinder</th>
<th>PDG-based</th>
<th>DECKARD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Type-3  Type-4
다른 도구들과 비교

**CCfinder**
- textual tokens

**PDG-based**
- program dependency graphs

**DECKARD**
- characteristic vectors

Type-3

Type-4
문제점?

- DiffCat은 자바프로그램만, MeCC은 C 프로그램만 지원
- MeCC은 함수 단위로만 유사 코드 창을 찾음
문제점?

• DiffCat은 자바프로그램만, MeCC은 C 프로그램만 지원
• MeCC은 함수 단위로만 유사 코드 쌍을 찾음

자바 분석기 위에 MeCC 방법을 적용하는 구현 중
Non-Essential Changes in Version Histories

David Kawrykow and Martin P. Robillard
McGill University
Montreal, Canada
{dkawry,martin}@cs.mcgill.ca

ABSTRACT
Numerous techniques involve mining change data captured in software archives to assist software engineering efforts, for example to identify components that tend to evolve together. We observed that many change-based approaches generally aim to operate on more meaningful representations of non-essential differences. We investigated the potential impact of non-essential differences in change history and their potential impact on change-based analyses.

Keywords
Mining software repositories, software change analysis, differencing algorithms

1. INTRODUCTION
Source code repository systems have been in use since the 1970s to keep track of the different versions of a software artifact and, by extension, of the changes made between versions [22]. Numerous techniques now involve mining change data captured in software repositories to mine change data for useful information, such as to predict defects in modules [10, 16], and to identify non-obvious relationships between code elements [8, 23, 25]. We refer to approaches operating on change data as change-based approaches. Given the growing importance of change analysis in software engineering, our long-term goal is to enable change-based approaches to incorporate information about the non-essential nature of changes into their analyses. With this information, change-based approaches will be able to more precisely select the individual low-level modifications that are most closely related to the high-level representations of development activity or effort.

We investigated the potential impact of non-essential differences on the abstracts that are typically analyzed by many change-based approaches. In particular, we sought to characterize the prevalence of non-essential differences in change history, and, to measure their impact on the code churn and method updates associated with code commits, two facets of code change that are considered in existing empirical studies involving change data [5, 16].

Analyzing change history to detect the kinds of non-essential differences mentioned above is a long-standing issue. An unnamed detection of non-essential differences requires both a characterization of structural changes occurring within statements and an analysis of their impact on the underlying system. In addition, to avoid reconstructing an entire program snapshot for every committed change, the impact of changes must be determined given only a change set, or group of files that were co-committed by a developer [26].