

Vaci:

# Variation Analysis of Context-Sharing Identifiers with Code Clones

Toshihiro Kamiya

Service Innovation Management Research Team

National Institute of Advanced Industrial Science and Technology, Japan

t-kamiya@aist.go.jp

# Naming, an old and new problem

Everyday developers give names to types, functions, variables, enums, ...

Adequate (descriptive, consistent) name is important.

- contributes to maintainability and understandability
- → naming convention

However, it's not always possible.

- Changelogs contain “rename”, “replace”...

Some tools have been proposed to support naming / renaming.

# Tool support

- Support for renaming
  - Name **restructuring** tool (overhauling names in a product)
    - B. Caprile and P. Tonella, “Restructuring Program Identifier Names”, *Proc. IEEE ICSM’00*, pp. 97-107, Oct. 2000.
  - Rename **refactoring** menu (such as Eclipse IDE)
  - “@deprecated” JavaDoc, **annotation** (Java)
  - Source **code converter** (Python 2.5 → 3.0)
- Support for checking name
  - Consistency checker (name  $\Leftrightarrow$  type)
    - F. Deißeböck and M. Pizka, “Concise and Consistent Naming”, *Proc. IWPC 2005*, pp. 97-106. May 2005.
    - D. Lawrie, et al, “Syntactic Identifier Conciseness and Consistency”, *Proc. IEEE SCAM 2006*, pp. 139-148, Sep. 2006.
  - Spell checker (such as Eclipse IDE)

# Tool Vaci

The tool collects and analyzes relations between each name and its contexts (i.e., code fragments where the name appears).

- Finding Variation of Names: to check consistency among the names that are used in the similar contexts.
- Tracking Change of Names: to track how names changes between versions.

# Snapshot #1, finding variation of names

The viewer works as Eclipse plugin.

It reads: the name “metricFile” and “cloneSetMetricFile” are used in the similar context.

I should rename the former name when I coded the latter...

The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Shows the project structure with packages like 'gemx', 'gemx.dia.ogs', and 'resources'.
- Code Editor:** Displays the source code for 'MainWindow.java'. A yellow callout points to a red circle in the toolbar, likely representing the viewer plugin. The code includes a method 'add\_file\_metrics()' with several lines of Java code.
- Problems/Declaration View:** Shows a list of identifiers found in the code, including 'metricFile', 'cloneSetMetricFile', and various methods like 'getSelection' and 'indexOf'.
- Vaci Search Dialog:** A dialog box titled 'Vaci search' is open, showing filtering options. The 'Filtering by Type' section includes checkboxes for 'nearly-identical', 'distinct-numbers', 'short-name', 'same-prefix', 'same-postfix', 'distinct-c', 'abbreviated', 'minority', and 'others'. The 'Find:' field is empty, and there are 'OK' and 'Cancel' buttons.

# Snapshot #2, tracking change of names

The analysis result is rendered into html document, including graphics (Scalable Vector Graphics).

It reads: the identifiers named "HWMODE\_A" is renamed to "IEEE80211\_BAND\_5GHZ".

Also, some identifiers named "MODE\_IEEE80211A" is renamed to "IEEE80211\_BAND\_5HZ." The others is renamed to "AR5K\_MODE\_11A".

The screenshot shows a Mozilla Firefox browser window with the address bar pointing to a local file. The main content area displays a list of identifiers and a translation map. The translation map shows a graph where 'HWMODE\_A' is renamed to 'IEEE80211\_BAND\_5GHZ' and 'MODE\_IEEE80211A' is renamed to 'AR5K\_MODE\_11A'. Below the graph, the edges (name changes) are listed: 'HWMODE\_A → IEEE80211\_BAND\_5GHZ'. At the bottom, there is a path to a file: 'c:\experiments\linuxkernel\_versions\linux-2.6.25\drivers\net\wireless\rt2x00\rt61pci:430 → c:\experiments\linuxkernel\_versions\linux-2.6.26\drivers\net\wireless\rt2x00\rt61pci:518'.

Firefox browser window showing a file:// path and a list of identifiers (e.g., #930 #935 #980 #1023 #1026 #1030 #1033 #1035 #1045 #1051).

**Type MtoN,single**

#8 #256 #257 #420 #421 #422 #423 #519 #623 #800  
 #809 #975 #1001 #1017 #1029 #1113 #1376 #1706 #1707 #1711  
 #1742

**Type MtoN,multiple**

**Translation map #17**

Graph

```

  graph LR
    HWMODE_A[x] --> IEEE80211_BAND_5GHZ
    MODE_IEEE80211A[x] --> AR5K_MODE_11A[@]
  
```

→ merge nodes

Edges (name changed)

HWMODE\_A → IEEE80211\_BAND\_5GHZ

c:\experiments\linuxkernel\_versions\linux-2.6.25\drivers\net\wireless\rt2x00\rt61pci:430 → c:\experiments\linuxkernel\_versions\linux-2.6.26\drivers\net\wireless\rt2x00\rt61pci:518

# Basic idea:

## “Identifiers share a context”

The cases where two identifiers of distinct names appear in the identical code fragments.


identifier

File.PathSeparator

identifier

File.PathSeparatorChar

context

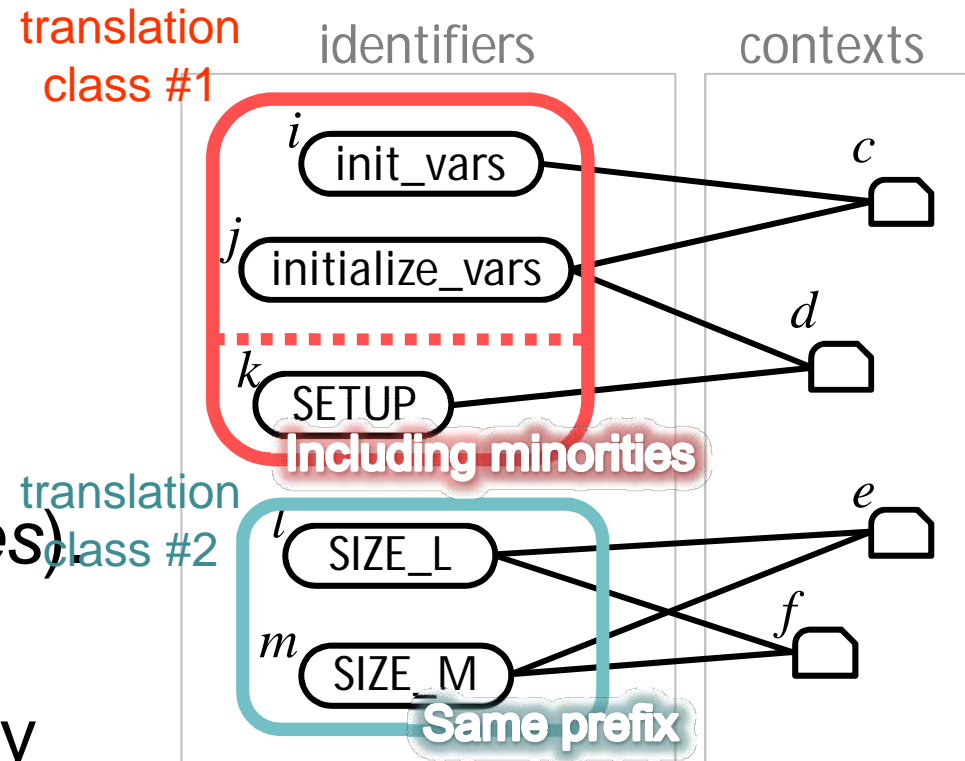
```
int pos = fpath.indexOf(  );
if (pos >= 0)
    fpath = fpath.substring(0, pos);
return fpath;
```

```
....
String getFirstPartOfPath(String fpath) {
    int pos = fpath.indexOf(File.pathSeparator);
    if (pos >= 0)
        fpath = fpath.substring(0, pos);
    return fpath;
}
....
```

```
....
String split1stPart(String fpath) {
    int pos = fpath.indexOf(File.pathSeparatorChar);
    if (pos >= 0)
        fpath = fpath.substring(0, pos);
    return fpath;
}
....
```

# Finding Variation of Names

1. Detect “context-sharing” identifier pairs.
2. Generate their transitive closures (*translation classes*)
3. Classify each translation class by the names of identifiers in the class



# Example from Linux Kernel

```
....
return -EINVAL;
}

result = i915_emit_irq(dev);

if (DRM_COPY_TO_USER(emit->irq_seq, &result, sizeof(int)) {
    DRM_ERROR("copy_to_user\n");
    return -EFAULT;
}

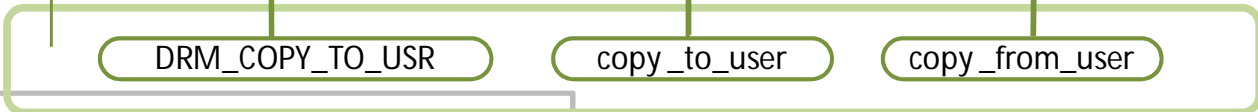
return 0;
}
...
```

```
....
return -EINVAL;
}

if (copy_to_user(param->value, &value, sizeof(int)) {
    DRM_ERROR("copy_to_user\n");
    return -EFAULT;
}

return 0;
}
```

```
if ( ($ ($, &$, sizeof(int)) ) {
    $("");
    return -$.
}
return 0;
```



```
....
if (db->sample_size == 16 && !mono && db->src_factor == 1) {
    /* no translation necessary, just copy
    */
    if (copy_to_user(userbuf, dmabuf, dmaaccount))
        return -EFAULT;
    return dmaaccount;
}
....
```

```
....
for (sample = 0; sample < num_samples; sample++) {
    if (copy_from_user(usersample, userbuf,
        db->user_bytes_per_sample)) {
        return -EFAULT;
    }

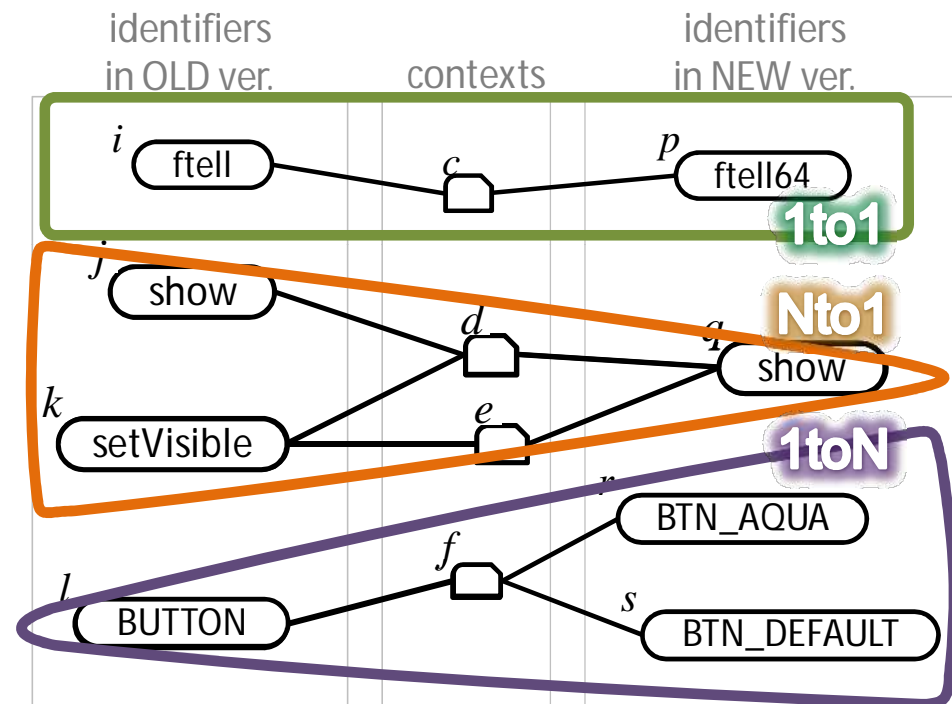
    for (i = 0; i < db->num_channels; i++) {

```

```
if ( ($ ($, $, $) ) {
    return -$.
}
```

# Tracking Change of Names

1. Detect context-sharing identifiers **between two versions.**
2. Generate their transitive closures (***translation maps***).
3. Classify each translation class **by number of identifiers**



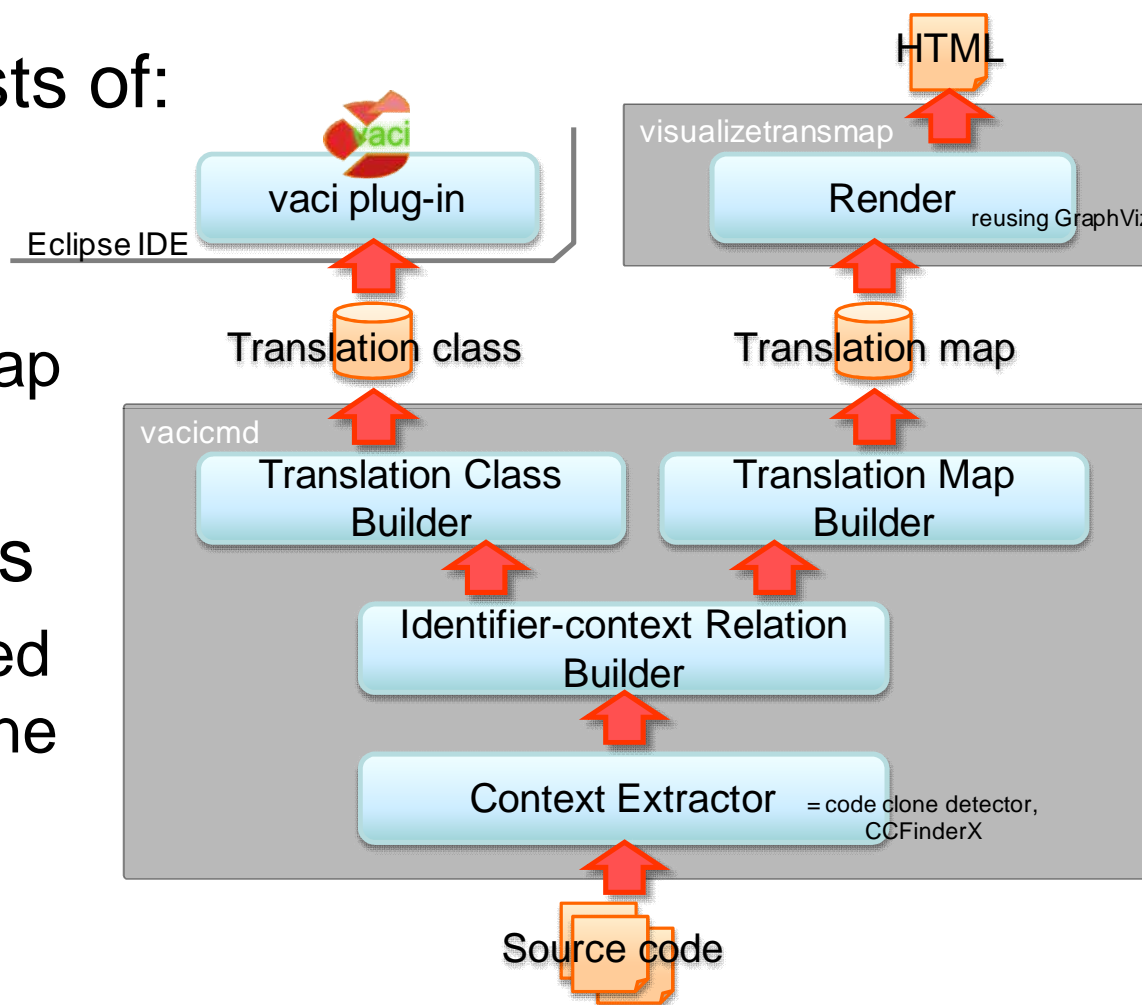
# Implementation

Tool Vaci consists of:

- vacicmd
- vaci plug-in
- visualizetransmap

Total 5k lines of  
Python/C++ lines

- Except for reused code from a clone detection tool “CCFinderX”



[www.ccfinder.net/vaci.html](http://www.ccfinder.net/vaci.html)

The latest version of Vaci (used in this presentation) will be available soon.

## Other useful materials

- Observations, theories
  - Derek M. Jones, The New C Standard, Sentence 792 (identifiers). Taken from <http://www.coding-guidelines.com/cbook/sent792.pdf>, 2008/09/11.
- Keyword → Code
  - Greg Little and Robert C. Miller, “Keyword Programming in Java”, Proc. the 22th IEEE/ACM Int’l Conf. Automated Software Engineering, pp. 84-93, 2007