Florian Deißenböck & Markus Pizka

# Concise and Consistent Naming

May 15th 2005

IWPC 05 St. Louis

# Outline

»The limits of my language mean the limits of my world.«
*Ludwig Wittgenstein (1889–1951)*

- Relevance of identifiers
- Naming Troubles
- Our approach
- Experiences
- Tool support
- Conclusion & Future Work

# Relevance of identifiers

- A program's *vocabulary* consists of its *identifiers*.
- Identifiers offer the most intuitive approach to program comprehension.
- About 70% of source code are identifiers.
- Obfuscators use *identifier scrambling* to make programs incomprehensible.

## Example Eclipse 3.0M7

- # Identifiers: 94.829
- # Words: 7.233

| Type | # | % | chars | % |
|---|---|---|---|---|
| Keywords | 967.665 | 11% | 4.650.273 | 13% |
| Delimiters | 4.096.112 | 47% | 4.096.112 | 11% |
| Operators | 531.444 | 6% | 669.932 | 2 % |
| Identifiers | 2.873.232 | 32% | 25.646.263 | 72% |
| Literals | 301.081 | 3% | 708.308 | 2% |
| Total | 8.769.534 | 100% | 35.770.888 | 100% |

Token-Analysis

# Naming troubles

- Identifiers in most programs are neither concise…
- …nor consistent

```
function mr_mr_1(mr, mr_1)
  if Null(mr) or Null(mr_1) then
    exit function
  end if
  mr_mr_1 = (mr - mr_1)
end function
```

1

```
int counter = getCounter();
  …
String counter = "first";
```

2

## Causes

- Identifiers can be choosen arbitrarily by developers.
- Identifiers elude automated analysis.
- Developers have only limited knowledge of *all* identifiers of a program.
- Identifiers are subject to decay during system evolution.

# Naming example

```
fct p = (seq m s) seq seq m :
  p1 (<>, <>, s)

fct p1 = (seq m t, seq m l, seq m r) seq seq m:
  if r == <> then <>
  elif  (rest(r) == <>) ∧ (l == <>) then <t ○ <first(r)>>
    else p1(t ○ <first(r)>, <>, l ○ rest(r)) ○ \
      p1(t, l ○ <first(r)>, rest(r))
      fi
  fi
```

- Identifiers provide information at the black-box level…
- …and assist building mental models.
- Identifiers are the the most basic form of documentation.

# A formal approach

## Premise

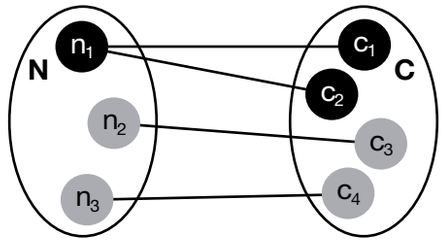
- A set of all relevant program concepts *C*.
- Concept *C* evolves over the program's lifetime.
- A set of names *N*.
- Assignment of names to concepts is modelled by relation *R*.
- Hyponymy and hypernymy relations are modelled by a partial order $<_A$.

$$permutation <_A transformation$$

# Consistency

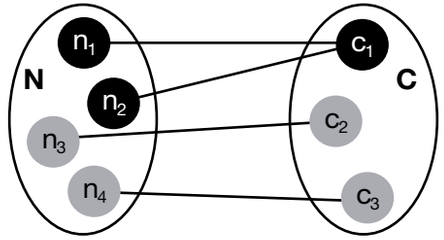
## Homonyms

- A *homonym* is a name referring to more than one concept.



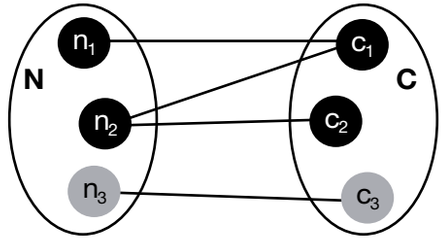

## Synonym

- Two names are *synonym* if they refer to the same concept.



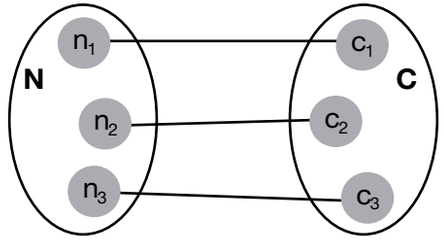

## Combination

- The combination of homonyms and synonyms increases the comprehension effort.




## Consistency

- A naming system is consistent if it contains neither homonyms nor synonyms.



# Conciseness

## Correctness

- A program element *p* implements concept *c*.
- The identifier of *p* is *correct* if it is the name of concept *c* or of a concept more general than *c*.
- »transformation« is a correct identifer for concept *permutation*.
- »storage« is no correct identifer for concept *permutation*.

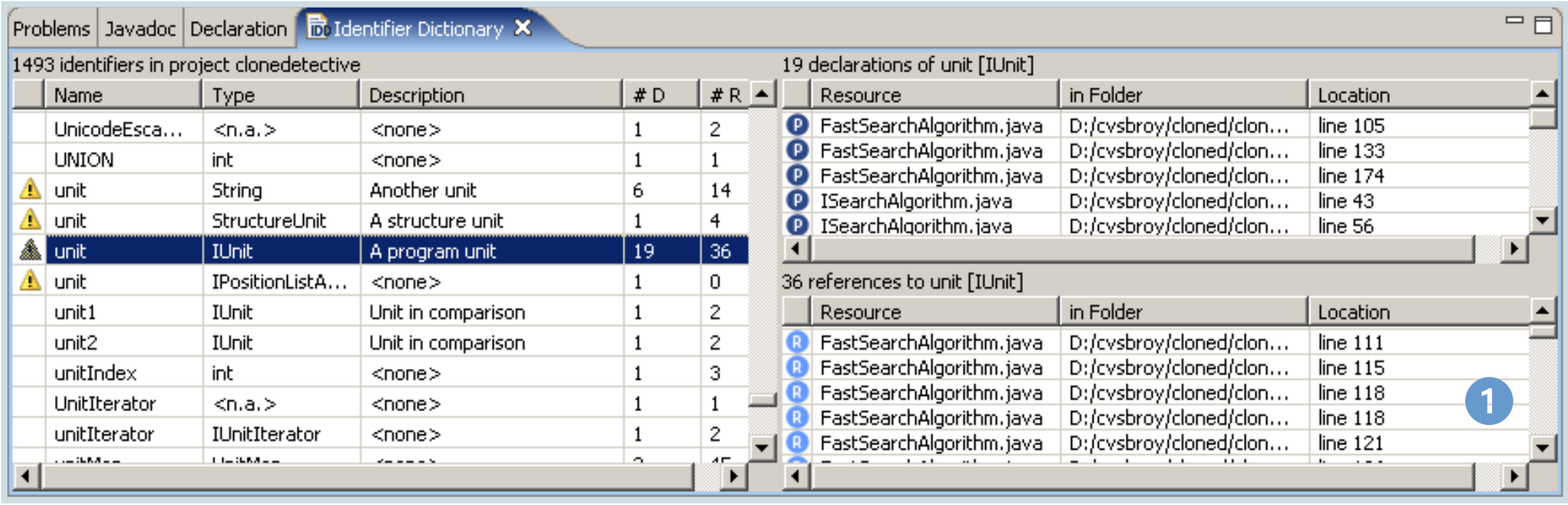
## Conciseness

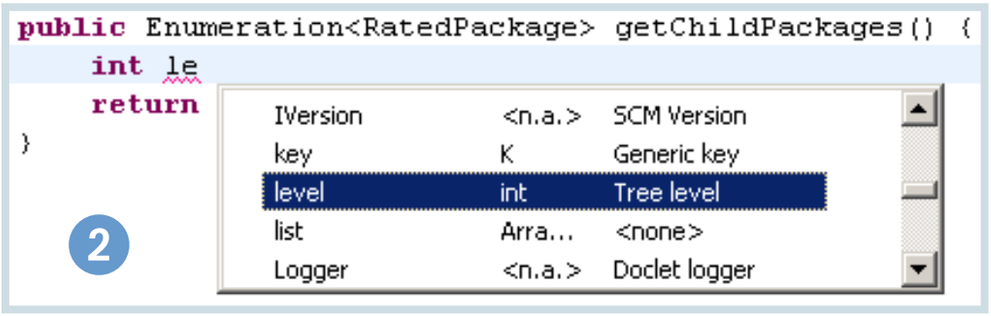
- A program element *p* implements concept *c*.
- The identifier of *p* is *concise* if it is the name of concept *c*.
- »permutation« is the only concise identifer for concept *permutation*.
- »transformation« is no concise identifer for concept *permutation*.

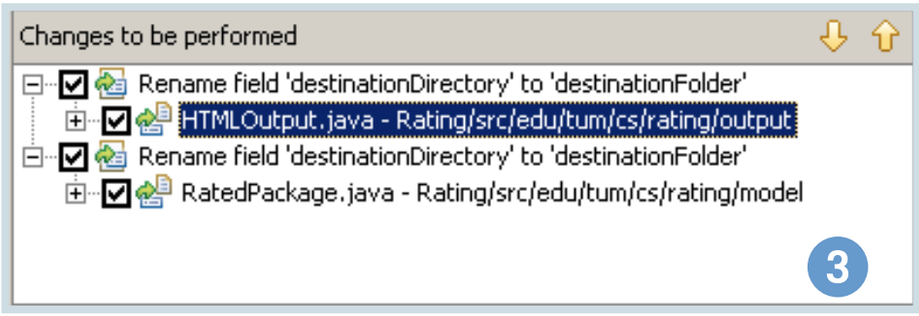
# Identifier Dictionary

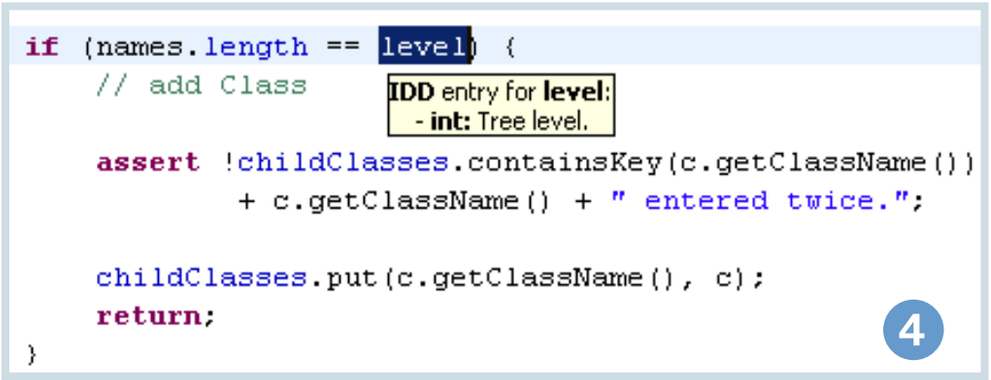
- *Identifier Dictionary (IDD)* stores all identifiers with a description.
- IDD is implemented as Eclipse plugin.
- Dictionary is stored in database or XML file.
- Dictionary is built incrementally by analyzing the AST.
- Warnings for typical naming problems.
- Hovers offer context-sensitive access to the dictionary.
- Auto-completion features help to avoid inconsistency.
- Global rename refactoring allows project-wide modification of identifiers.

# IDD in action




1. main view
2. auto-completion
3. refactoring preview
4. hover

# Experiences

- Tool usage fostered awareness for naming issues.
- Students developed identifier-guided strategies for reverse engineering tasks.
- Global rename refactoring was percieved as powerful feature to cure naming problems.
- Change tracking of the identifier dictionary helped to find potential problems.
- Current implementation doesn‘t regard names of non-source-code entities.

▶ Too early to draw a final conclusion but results are very encouraging.

# Conclusion & Future Work

## Conclusions

- Formal model allows thorough analysis of naming problems and…
- …increases awareness of naming issues.
- Identifier dictionary allows to apply the model.
- Application must be supported by an appropriate process.

## Future work

- Compound identifiers
  - Extension of the model to describe compound identifiers.
  - Definition of rules for identifier composition.
  - Improve IDD for better compound handling.
- Leverage ontologies for more sophisticated automatic analysis.