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

#### Token-Analysis

<table>
<thead>
<tr>
<th>Type</th>
<th>#</th>
<th>%</th>
<th>chars</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keywords</td>
<td>967,665</td>
<td>11%</td>
<td>4,650,273</td>
<td>13%</td>
</tr>
<tr>
<td>Delimiters</td>
<td>4,096,112</td>
<td>47%</td>
<td>4,096,112</td>
<td>11%</td>
</tr>
<tr>
<td>Operators</td>
<td>531,444</td>
<td>6%</td>
<td>669,932</td>
<td>2%</td>
</tr>
<tr>
<td>Identifiers</td>
<td>2,873,232</td>
<td>32%</td>
<td>25,646,263</td>
<td>72%</td>
</tr>
<tr>
<td>Literals</td>
<td>301,081</td>
<td>3%</td>
<td>708,308</td>
<td>2%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>8,769,534</td>
<td>100%</td>
<td>35,770,888</td>
<td>100%</td>
</tr>
</tbody>
</table>
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

Causes

- Identifiers can be chosen 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

\[
\text{fct } p = (\text{seq } m \ s) \ \text{seq seq } m : \\
p1 (\langle\rangle, \langle\rangle, s)
\]

\[
\text{fct } p1 = (\text{seq } m \ t, \ \text{seq } m \ l, \ \text{seq } m \ r) \ \text{seq seq } m : \\
\text{if } r == \langle\rangle \ \text{then } \langle\rangle \\
\text{elif } (\text{rest}\(r) == \langle\rangle) \land (l == \langle\rangle) \ \text{then } \langle t \circ \langle\text{first}\(r)\rangle\rangle \\
\text{else } p1(t \circ \langle\text{first}\(r)\rangle, \langle\rangle, l \circ \text{rest}\(r)) \circ \backslash \\
\quad p1(t, l \circ \langle\text{first}\(r)\rangle, \text{rest}\(r)) \\
\text{fi} \\
\text{fi}
\]

- Identifiers provide information at the black-box level…
- …and assist building mental models.
- Identifiers are 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 $<_A$.

\[
\text{permutation} <_A \text{transformation}
\]
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.
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 identifier for concept *permutation*.
  - »storage« is no correct identifier 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 identifier for concept *permutation*.
  - »transformation« is no concise identifier 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 perceived 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.