Mining the Lexicon Used by Programmers during Software Evolution

ITC
ISTITUTO TRENTINO
DI CULTURA
Dal 1° marzo 2007
FONDAZIONE
BRUNO KESSLER
irst CENTRO PER LA RICERCA
SCIENTIFICA E TECNOLOGICA

Giuliano Antoniol, **Yann-Gaël Guéhéneuc**,
Ettore Merlo, **Paolo Tonella**
From:
École Polytechnique de Montréal, Canada
University of Montreal, Canada
FBK-IRST, Trento, Italy

Is this one the class you are looking for?

```
public class T01<T02,T03> extends T04<T02,T03>
  implements T05<T02,T03>, T06, T07 {

  public T03 m01(T02 x01, T03 x02) {
    if (x01 == null)
      return m02(x02);
    int x03 = m03(x01.m04());
    int x04 = m05(x03, x05.x06);
    for (T08<T02,T03> x07 = x08[x04]; x07 != null; x07 = x07.x09) {
      T09 x10;
      if (x07.x11 == x03 && ((x10 = x07.x12) == x01 || x01.m06(x10))) {
        T03 x13 = x07.x14;
        x07.x14 = x02;
        x07.m07(this);
        return x13;
      }
    }
    x15++;
    m08(x03, x01, x02, x04);
    return null;
  }
}
```

Is this one the class you are looking for?

```
public class HashMap<K,V> extends AbstractMap<K,V>
  implements Map<K,V>, Cloneable, Serializable {

  public V put(K key, V value) {
    if (key == null)
      return putForNullKey(value);
    int hash = hash(key.hashCode());
    int i = indexFor(hash, table.length);
    for (Entry<K,V> e = table[i]; e != null; e = e.next) {
      Object k;
      if (e.hash == hash && ((k = e.key) == key || key.equals(k))) {
        V oldValue = e.value;
        e.value = value;
        e.recordAccess(this);
        return oldValue;
      }
    }
    modCount++;
    addEntry(hash, key, value, i);
    return null;
  }
}
```

Self-documenting identifiers

Good identifiers:
- ➢ provide concise clues on the semantics of labeled entities;
- ➢ save programmers from reading the entire code segment;
- ➢ speed up knowledge acquisition;
- ➢ support program understanding (code queries, grep, etc.).

To some extent, we know how the structure of a program evolve.

How does the lexicon of identifiers evolve?

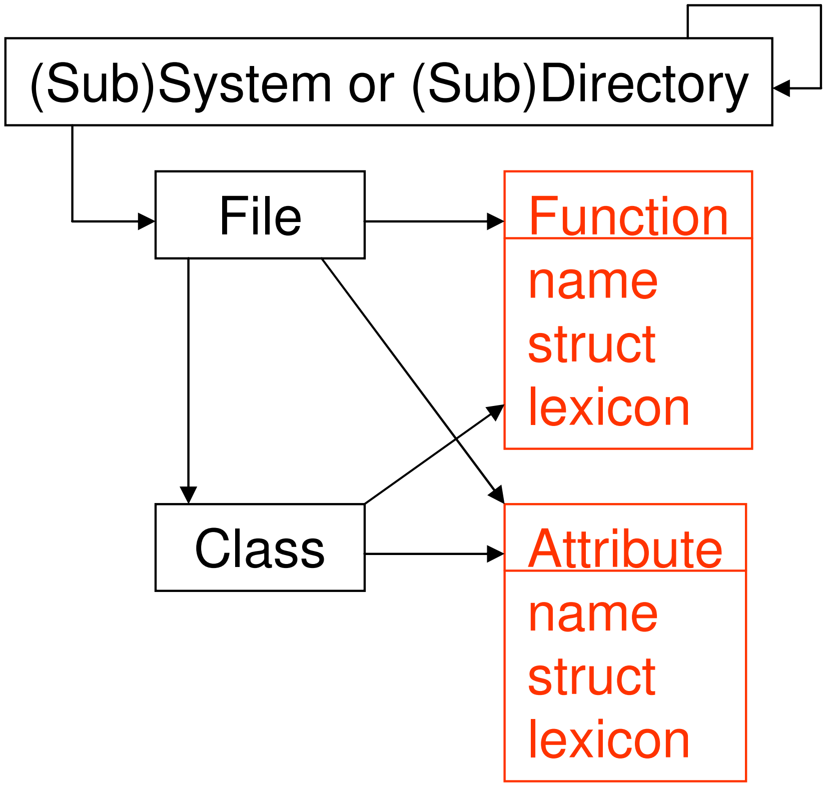
Corollary: When we teach programming, we should never let our students use names such as **`foo`** or **`bar`** (**`pippo, pluto`**) for any program entity.

Research questions

RQ1: How does the stability of the lexicon of identifiers compare to the stability of the program structure as the program evolves?

RQ2: What is the frequency of changes to program entities (in particular renaming) due to identifier refactoring?

Data model



```
class HashMap<K, V> {
  Entry<K, V> table[];
  V put(K key, V value) {…}
}
```

Full lexicon:
<hash, map, table, put, key, value>

Function
Name: put
Struct: <10, 1, 2, 31, 0, 0, 24>
Lexicon: <0, 0, 0, 1, 1, 1>

Attribute
Name: table
Struct: <1, …>
Lexicon: <0, 0, 1, 0, 0, 0>

Stability metrics

For leaf entities, **cosine similarity:**

StructSim(Ei, Ej) = <struct(Ei), struct(Ej)> / |struct(Ei)| |struct(Ej)|

LexicalSim(Ei, Ej) = <lexicon(Ei), lexicon(Ej)> / |lexicon(Ei)| |lexicon(Ej)|

Function

Name: put
Struct: <10, 1, 2, 31, 0, 0, 24>
Lexicon: <0, 0, 0, 1, 1, 1>

Function

Name: put'
Struct: <11, 2, 1, 30, 0, 0, 22>
Lexicon: <0, 0, 0, 1, 1, 1>

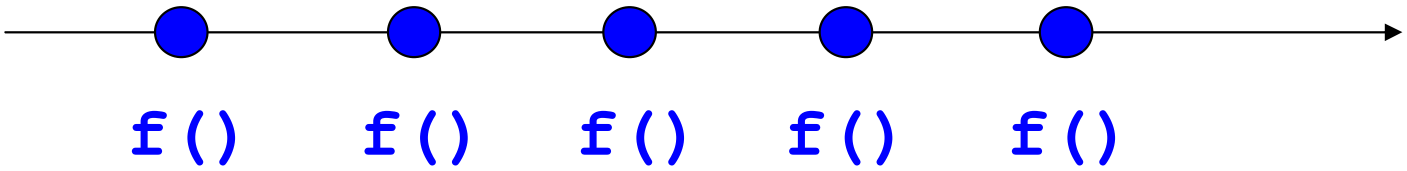
StructSim(put, put') = 0.998
LexicalSim(put, put') = 1

For container entities, **average similarity**

Similarity between corresponding entities in the history ► entity traceability required!

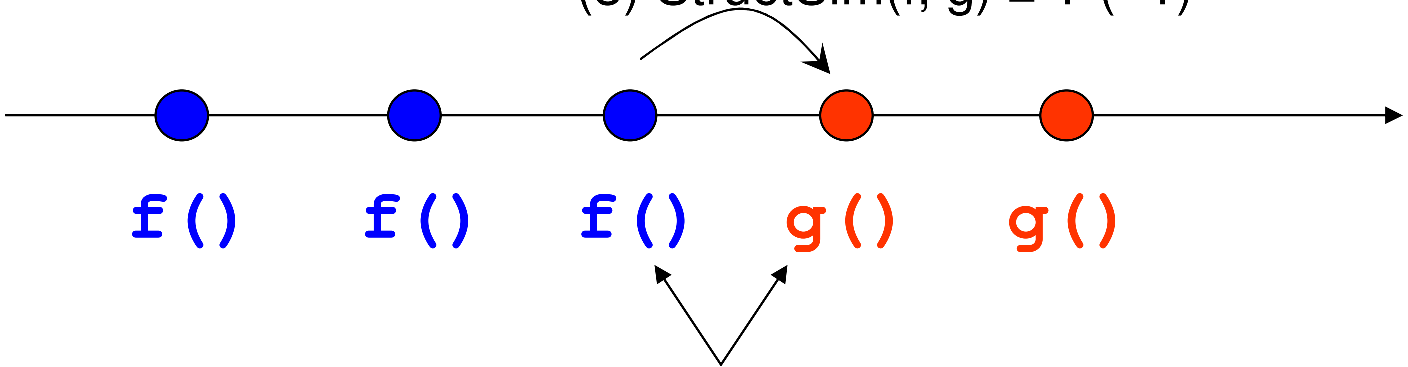
Entity traceability

By name:



By structure:

(1) Traceability by name fails
(2) There is no entity g() in previous release
(3) StructSim(f, g) ≥ T (=1)



Renaming detected!

Metrics and analysis

RQ1 *(struct vs. lexicon evolution):*

Null hypothesis: there is no statistically significant difference between the probability distribution of lexical vs. structural stability.
Statistical test: non-parametric Wilcoxon paired test.

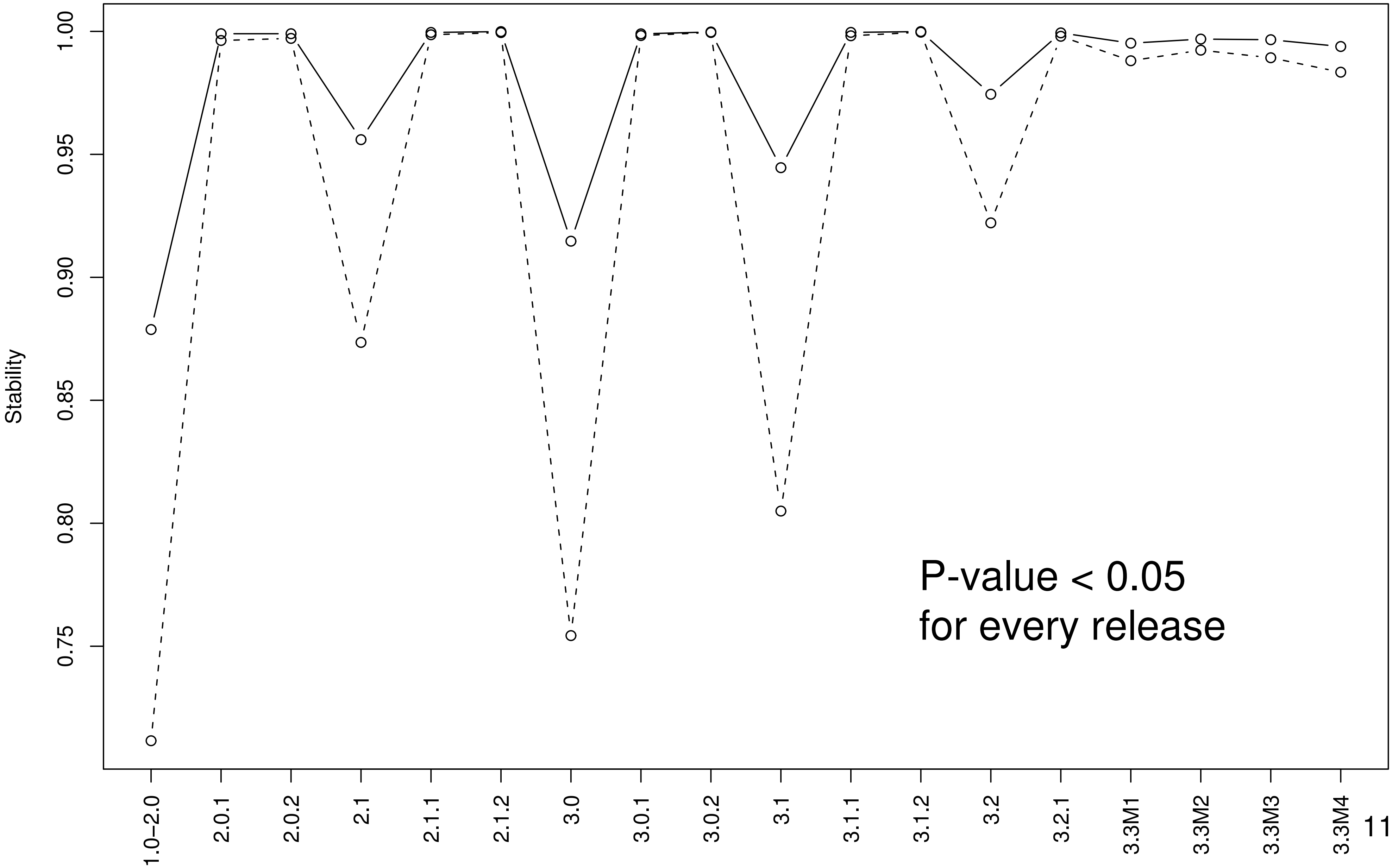
RQ2 (*frequency of renamings*):

RenFreq = DetectedRenamings / Total Entities

Subject systems

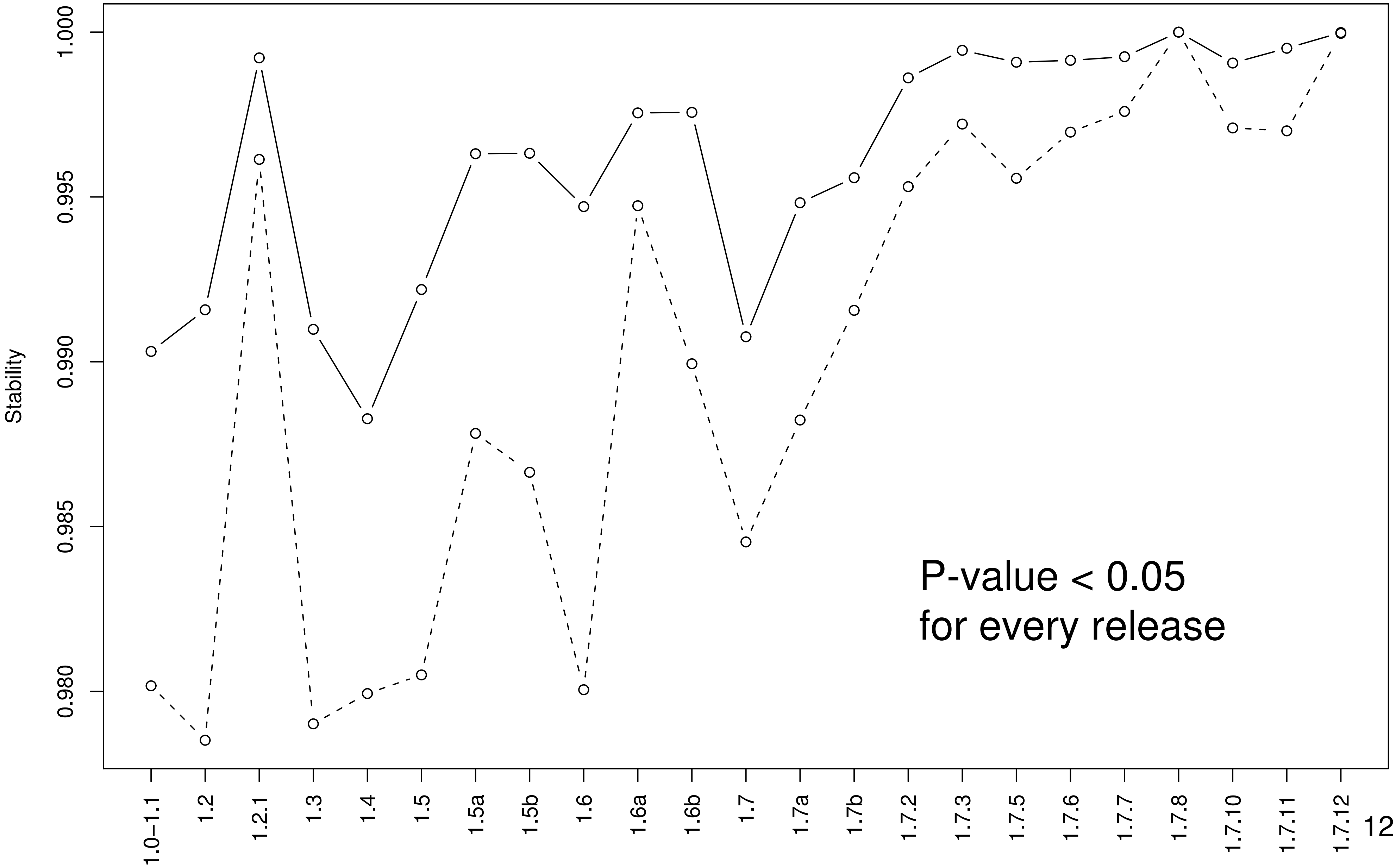
System	Language	Size	Versions	Identifiers
Eclipse	Java	2.9 MLOC	19	124187
Mozilla	C++	4.4 MLOC	24	55244
CERN/Alice	C++	0.825 MLOC	13	9002

Stability plot: Eclipse



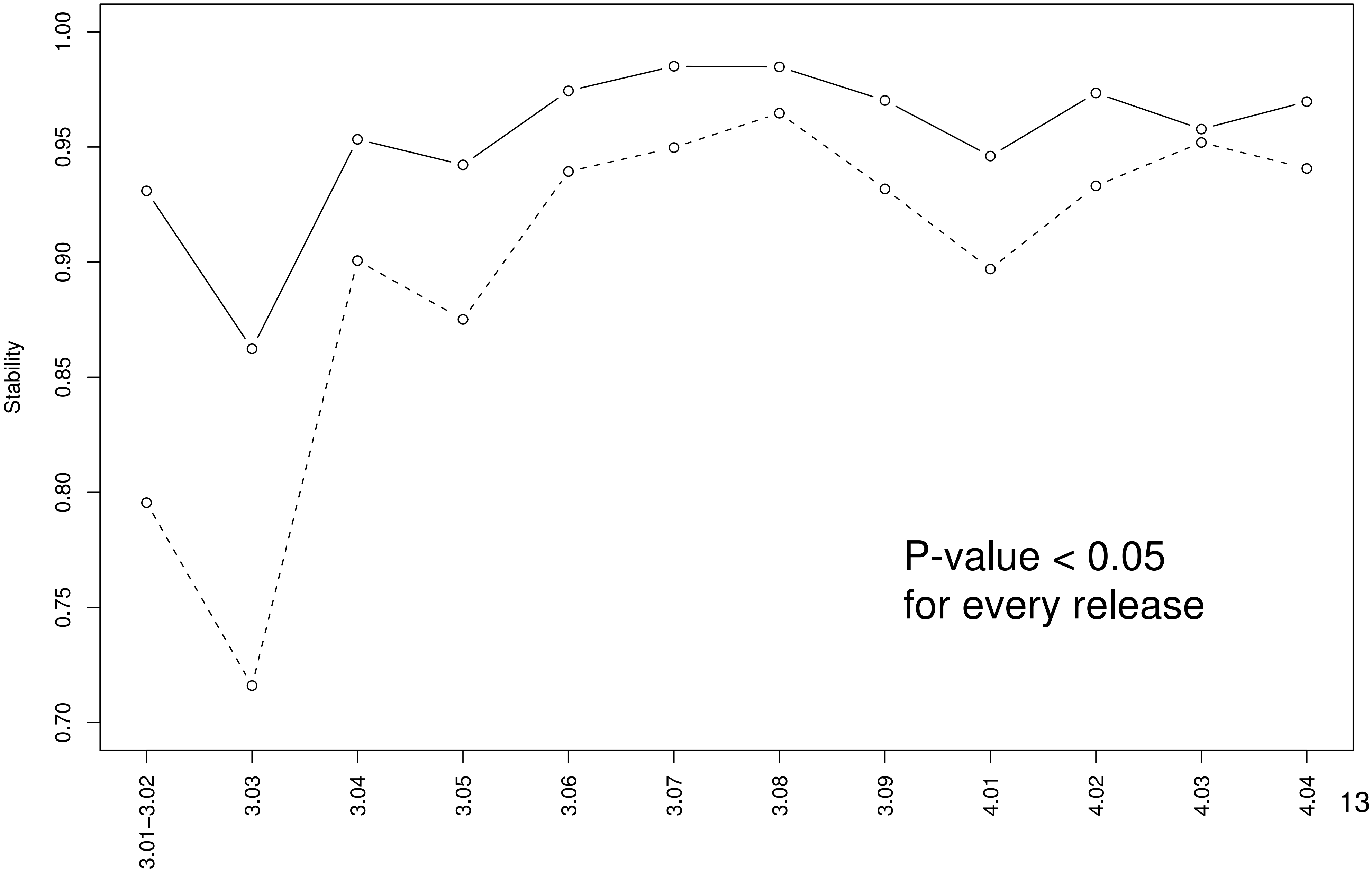
P-value < 0.05
for every release

Stability plot: Mozilla



P-value < 0.05
for every release

Stability plot: Alice



P-value < 0.05
for every release

Renaming

Eclipse (Java):	AvgRenFreq = 7 / 106760 = 0.000065
Mozilla (C++):	AvgRenFreq = 0 / 51981 = 0
Alice (C++):	AvgRenFreq = 0 / 6736 = 0

Summary

RQ1 *(struct vs. lexicon evolution):*

➢ *Lexical and structural changes have different distributions over time; they probably obey different rules.*
➢ *Lexicon is always more stable than structure.*
➢ *Both structural and lexical stabilities tend to increase over time and tend to have correlated instabilities.*

RQ2 (*frequency of renamings*):

➢ *Renamings are rare during the evolution of a software system.*

Discussion (our interpretations)

- A different change process holds for lexicon and structure.
- Programmers are generally reluctant to change the lexicon. Some possible reasons:
  - Optimistically, there is no need to do it (domain perfectly modeled by lexicon).
  - High cognitive burden associated with this kind of change.
  - No dedicated tool available.
- The development environment seems to have an influence on the evolution of the lexicon. A renaming tool available in the IDE may help (Java vs. C++ in our study).
  - Other tools that may help: glossaries, cross-referencing tools, abbreviation expansion tools, documentation tools (possible using ontologies).

Corollary: A program written with a bad lexicon (**`foo`**, **`bar`**, **`pippo`**, **`pluto`** and the like) tends to keep its poor identifiers forever.
Programmers must adapt to them; the inverse rarely happens.

Conclusions and future work

RQ1: The lexicon is more stable than the structure.

RQ2: Identifier restructuring is rare.

Drafting a future work agenda:

- The lexicon of a program represents a substantial investment for a company.
- However, almost no support is available to preserve and increase such value over time.
- Research on techniques and tools for program lexicon analysis and manipulation is strongly needed.