FONDAZIONE
BRUNO KESSLER

Code Quality from the Programmer's Perspective

Paolo Tonella

Fondazione Bruno Kessler

Trento, Italy

Two perspectives on code quality

Does the software correctly implements requirements? Is it easy to use? Does it crash?

How is the code organized? Where is <some> functionality implemented?

External (user's): *what?*

Internal (programmer's): *how?*

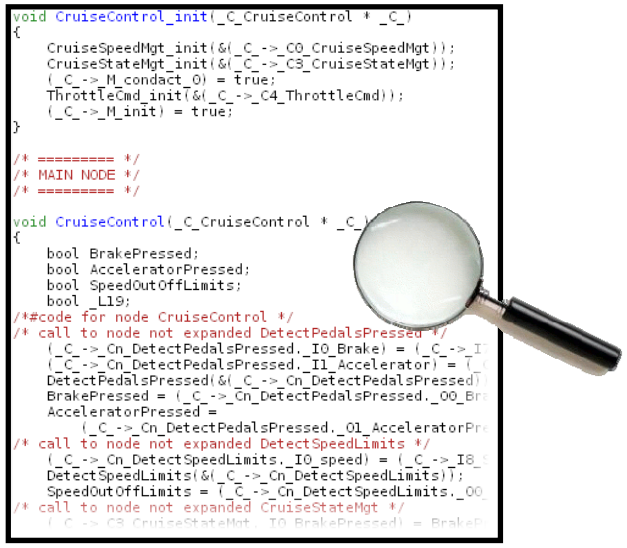
FONDAZIONE
BRUNO KESSLER

User's frustration

- "after clicking all options and navigating all menus I still cannot find out how to achieve <this>"
- "why do I get this error message? I did nothing wrong – stupid software!"

FONDAZIONE BRUNO KESSLER

Programmer's frustration



```
void CruiseControl_init(_C_CruiseControl * _C_)
{
    CruiseSpeedMgt_init(&(_C_->_C0_CruiseSpeedMgt));
    CruiseStateMgt_init(&(_C_->_C3_CruiseStateMgt));
    (_C_->_M_condact_0) = true;
    ThrottleCmd_init(&(_C_->_C4_ThrottleCmd));
    (_C_->_M_init) = true;
}

/* ========= */
/* MAIN NODE */
/* ========= */

void CruiseControl(_C_CruiseControl * _C_
{
    bool BrakePressed;
    bool AcceleratorPressed;
    bool SpeedOutOffLimits;
    bool _L19;
/*#code for node CruiseControl */
/* call to node not expanded DetectPedalsPressed */
    (_C_->_Cn_DetectPedalsPressed._I0_Brake) = (_C_->_I
    (_C_->_Cn_DetectPedalsPressed._I1_Accelerator) = (_
    DetectPedalsPressed(&(_C_->_Cn_DetectPedalsPressed
    BrakePressed = (_C_->_Cn_DetectPedalsPressed._O0_Br
    AcceleratorPressed =
        (_C_->_Cn_DetectPedalsPressed._O1_AcceleratorPr
/* call to node not expanded DetectSpeedLimits */
    (_C_->_Cn_DetectSpeedLimits._I0_speed) = (_C_->_I8_
    DetectSpeedLimits(&(_C_->_Cn_DetectSpeedLimits));
    SpeedOutOffLimits = (_C_->_Cn_DetectSpeedLimits._O0_
/* call to node not expanded CruiseStateMgt */
    (_C_->_C3_CruiseStateMgt._I0_BrakePressed) = BrakeP
```

- "I have to add this function, but it's impossible to find a single place where the addition should go in the intricate code structure"
- "the program produces the wrong output, but after several hours browsing the code I cannot identify which statements are guilty"

FONDAZIONE
BRUNO KESSLER

Industrial context

- Customer's needs and domain are only partially known.
- There is little time to improve code structure and internal quality.
- Most quality improvement effort goes in checking the implementation against the requirements, little goes on internal quality improvement (no time available for that).

FONDAZIONE BRUNO KESSLER

Open source context

- Users and developers are often the same persons.
- Domain and requirements are well known, so they need no explicit model.
- Many developers are supposed to contribute, so the code should be easy to read and modify.
- Code inspections happen all the time.

FONDAZIONE BRUNO KESSLER

What about physics software?

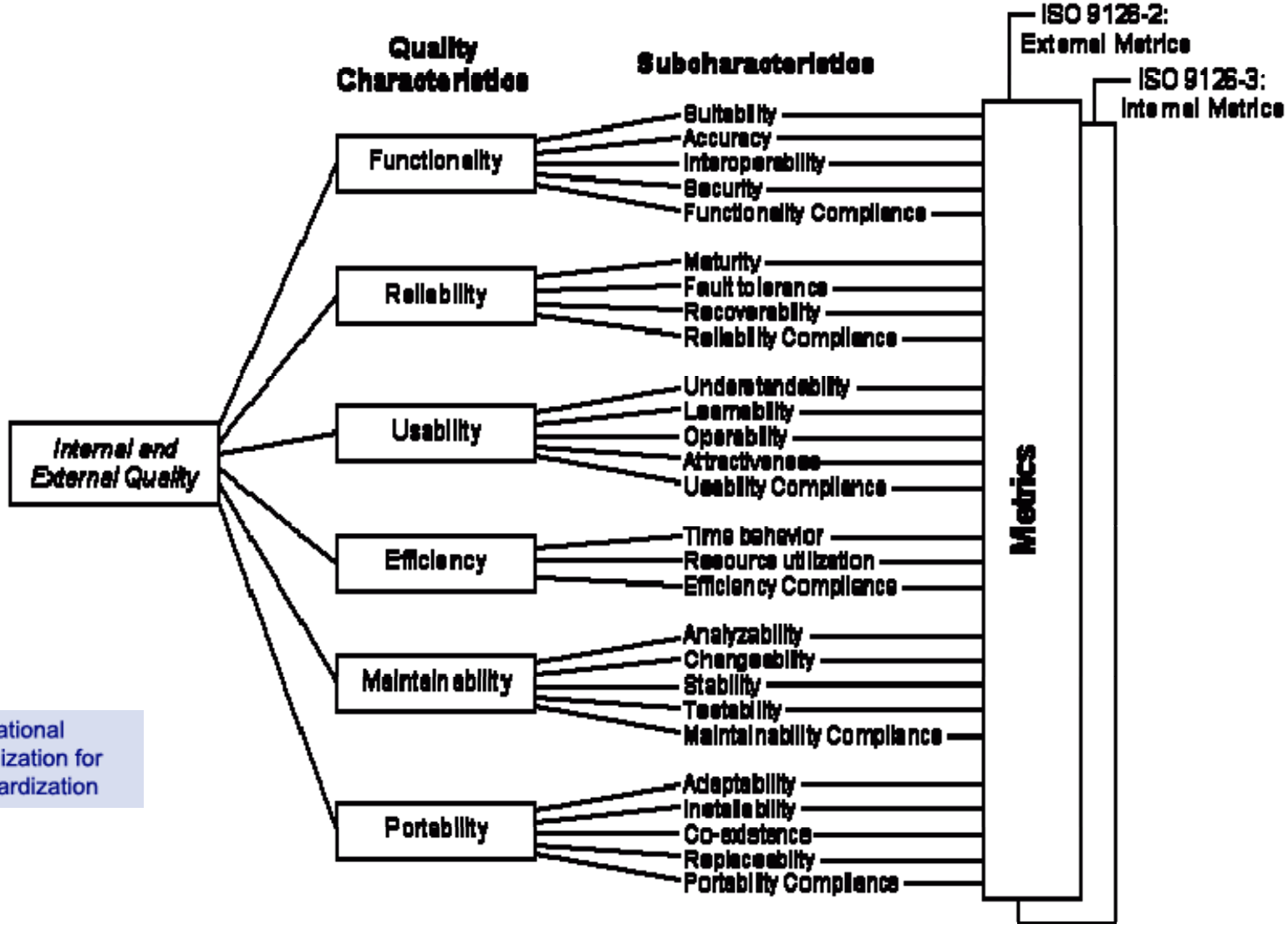
- Developers are typically users.
- Domain is perfectly known.
- Many contributors and advanced users are spread all over the world.
- Hence, internal quality is probably the major quality goal; external quality descends quite “naturally”.

A Large Ion Collider Experiment

European Organisation for Nuclear Research

FONDAZIONE
BRUNO KESSLER

Quality models (ISO 9126)



ISO International Organization for Standardization

FONDAZIONE BRUNO KESSLER

Internal code quality

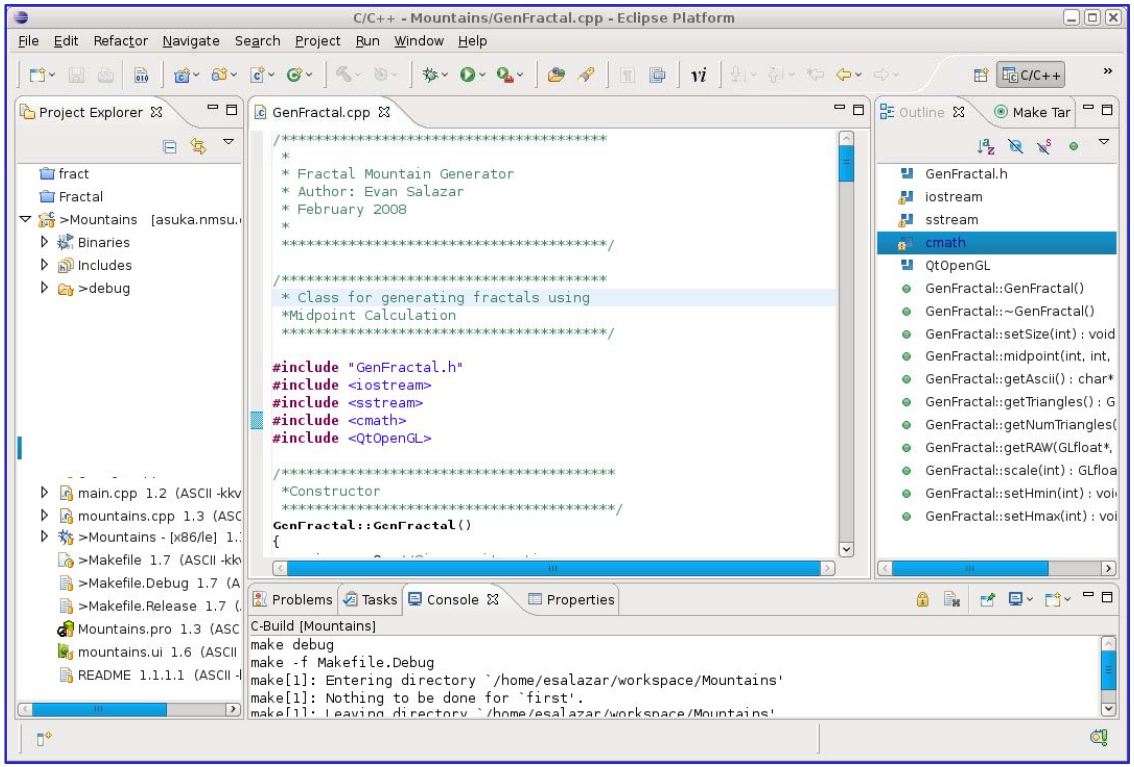
- **Metrics**: coupling, cohesion and other (DIT, CBM, LCM, etc.)
- **Code smells** and code refactoring: possibility to apply standard, known solutions (e.g., design patterns) and improvements.
- **Coding conventions**: coding rules, style rules, etc.

Internal quality still ignores the central role of **natural language** in programs.

Programmer's weapons

Let us consider the programmer's perspective, when executing a typical programming task (add function, fix bug, etc.):

- **Text processing**: Grep or CTRL-F.
- **IDE navigation**: Structure view, call graph, debugger, design diagrams, etc.



FONDAZIONE BRUNO KESSLER

Clues from natural language

- Available tools do not take advantage of natural language clues.
- Quality of natural language in the code is currently neglected.

FONDAZIONE BRUNO KESSLER

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;
  }
}
```

FONDAZIONE BRUNO KESSLER

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;
  }
}
```

FONDAZIONE BRUNO KESSLER

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.).

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

To some extent, we know a lot about the structure of a program.

What do we know about the lexicon of programs?

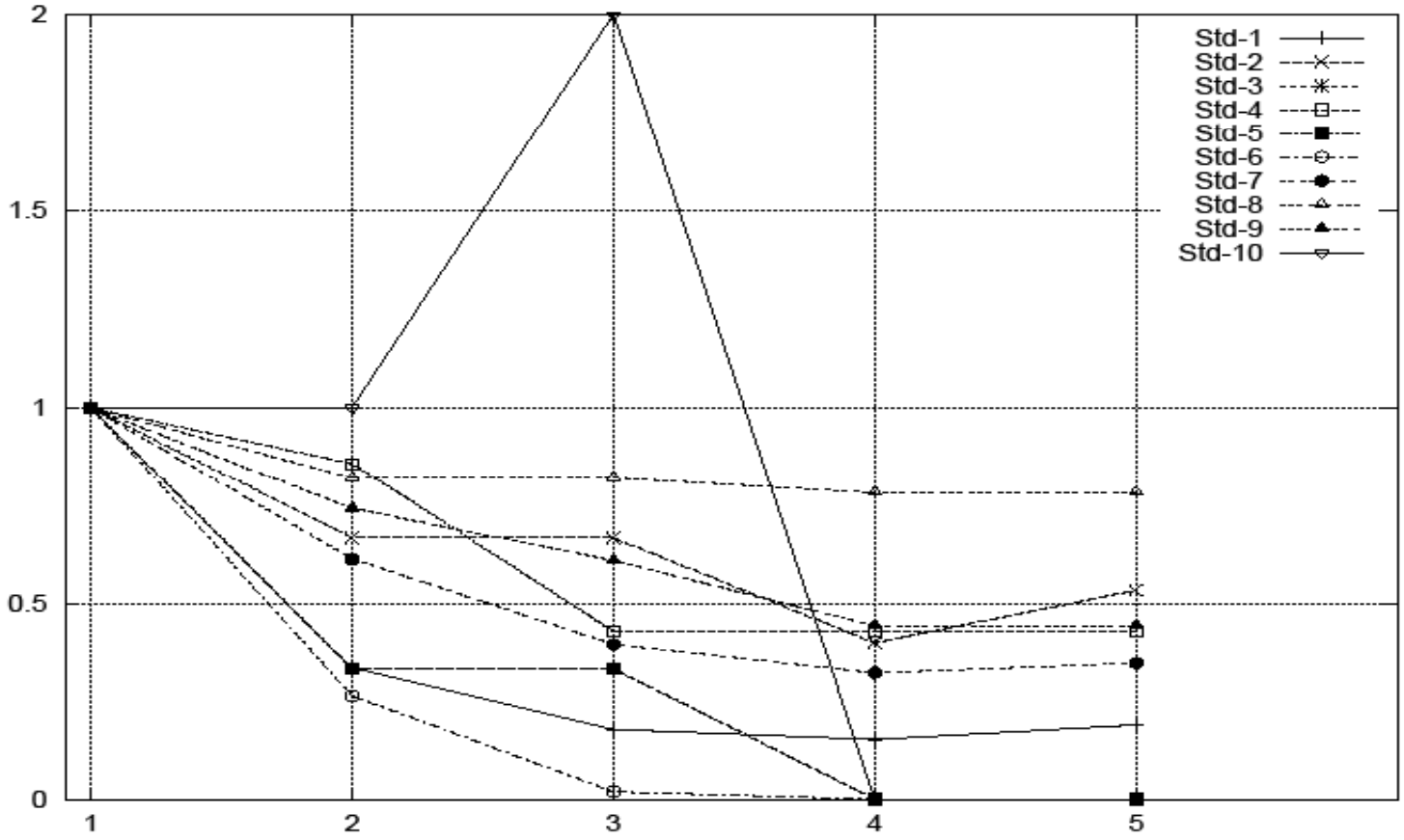
FONDAZIONE BRUNO KESSLER

Our experience with CERN/Alice

- Developed RuleChecker in 1999; NewRuleChecker delivered in 2008, based on srcML.
- Verified internal quality properties about structure, code organization, style, etc.
- Identified sub-systems with poor internal quality and enforced their improvement.
- Programmers felt the eye of RuleChecker over their code, so they probably paid more attention to coding rules and guidelines.

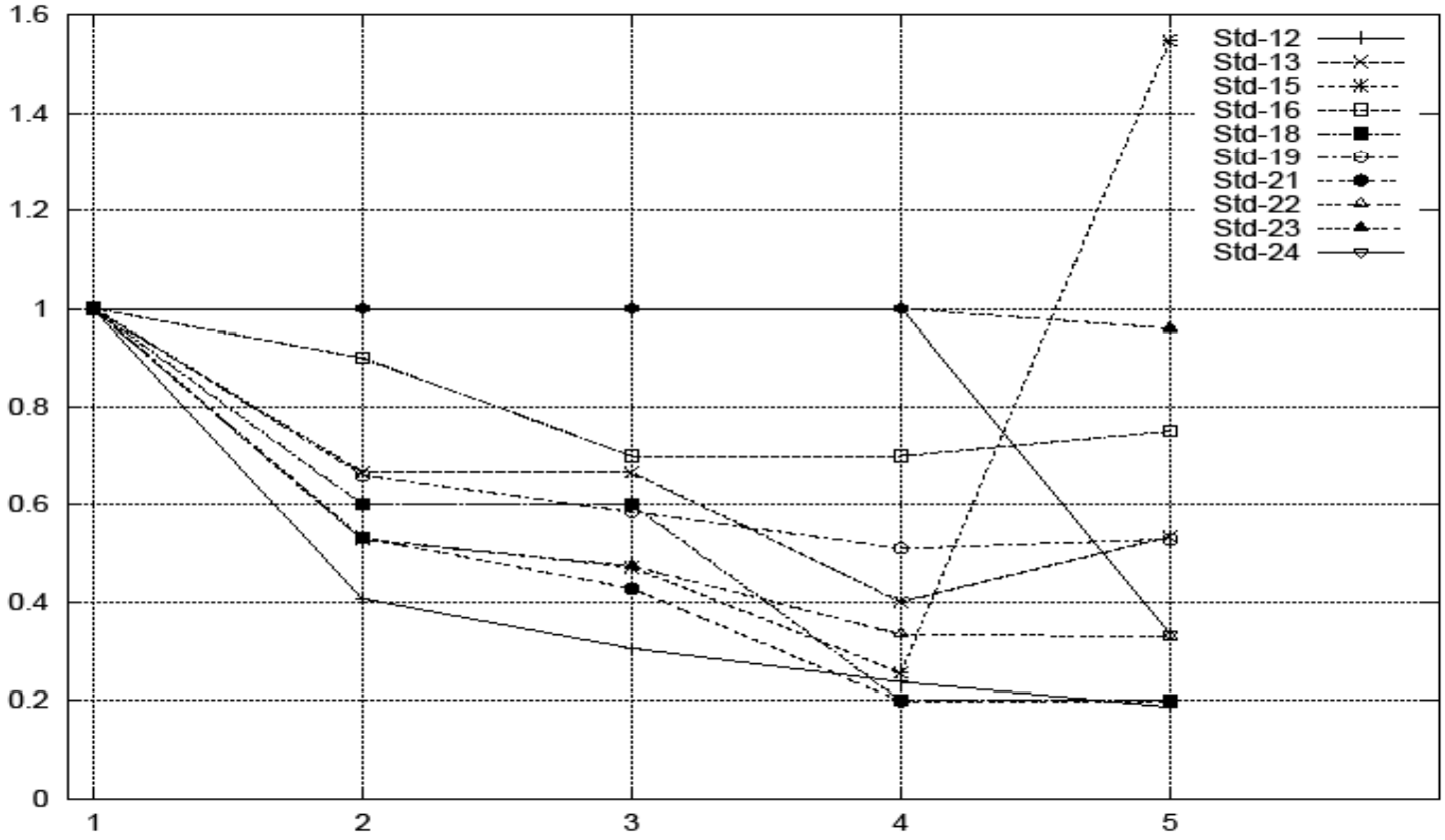
FONDAZIONE
BRUNO KESSLER

Naming rules



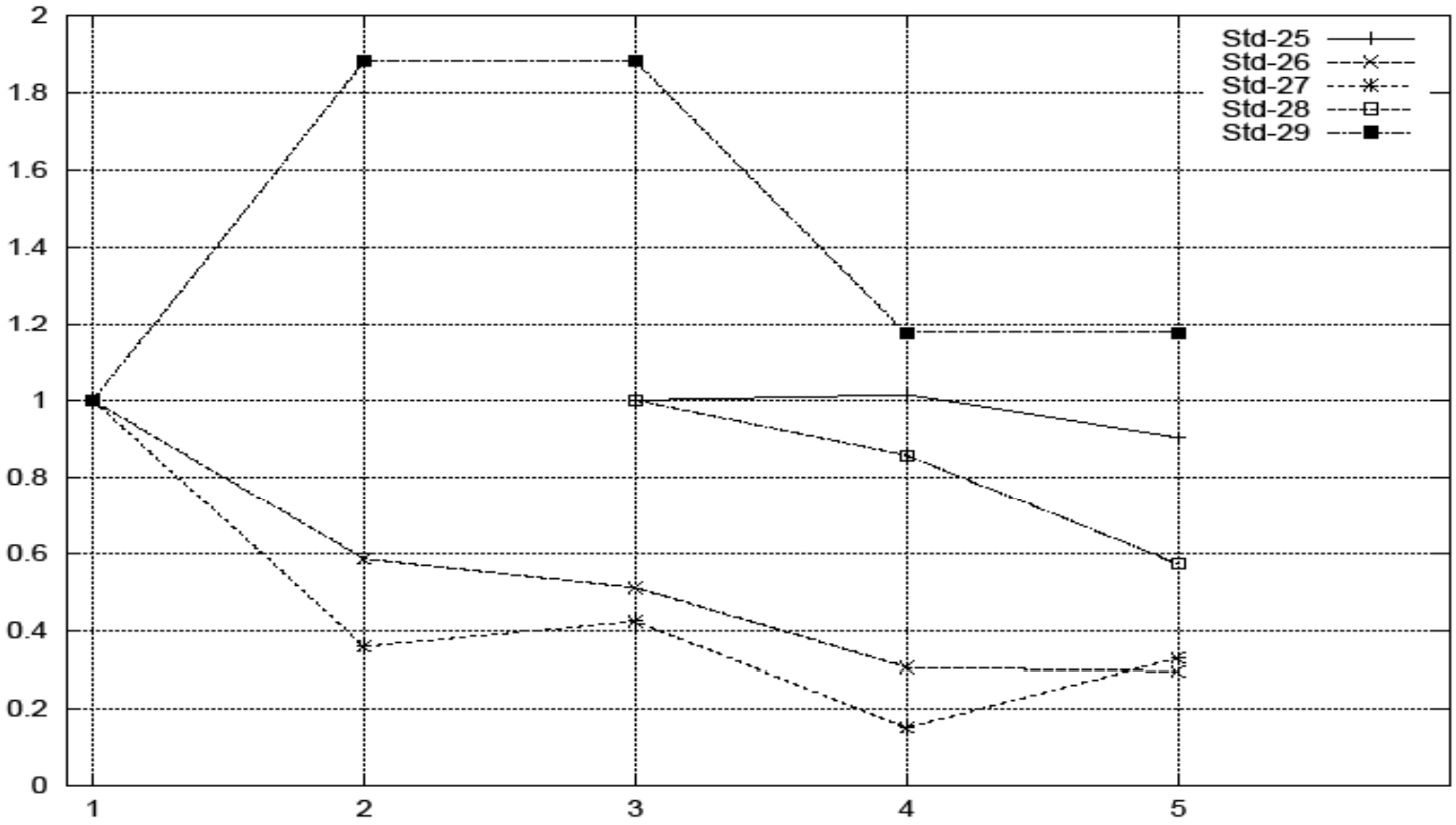
FONDAZIONE
BRUNO KESSLER

Coding rules



FONDAZIONE
BRUNO KESSLER

Style rules



FONDAZIONE
BRUNO KESSLER

Results

- Successfully improved internal quality indicators (metrics, code smells, coding conventions), mostly related to the internal code structure.
- Not yet considered the quality of the informal information embedded in the code: quality of programmer's lexicon and its consistent usage.

FONDAZIONE BRUNO KESSLER

Empirical studies on CERN/Alice code

- **Empirical study #1:** Stability of lexicon, compared to structure (ICSM 2007).
- **Empirical study #2:** Delta analysis: why and how is the lexicon evolved? (paper under review, 2008).

FONDAZIONE
BRUNO KESSLER

Empirical Study #1

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?

FONDAZIONE BRUNO KESSLER

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>

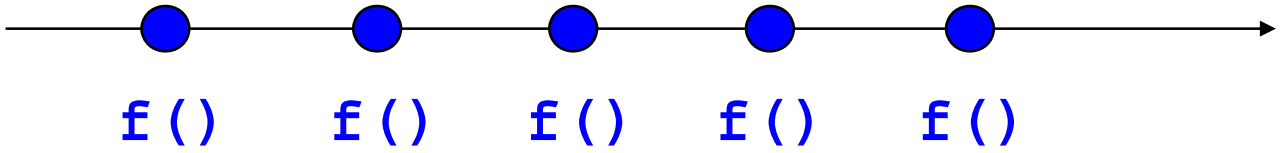
StructSim(put, put') = 0.998
LexicalSim(put, put') = 1

For container entities, **average similarity**

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

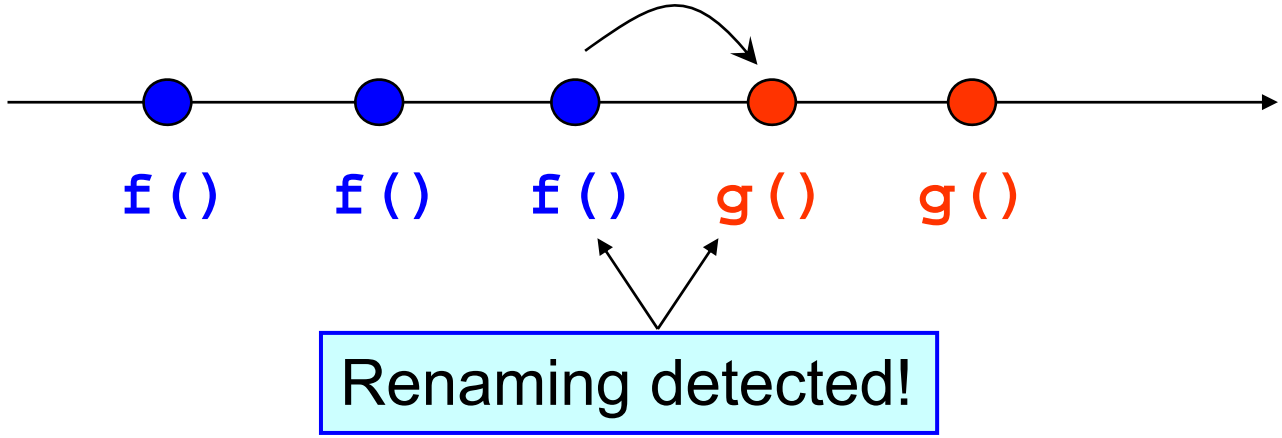
Entity traceability

By name:



By structure:

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

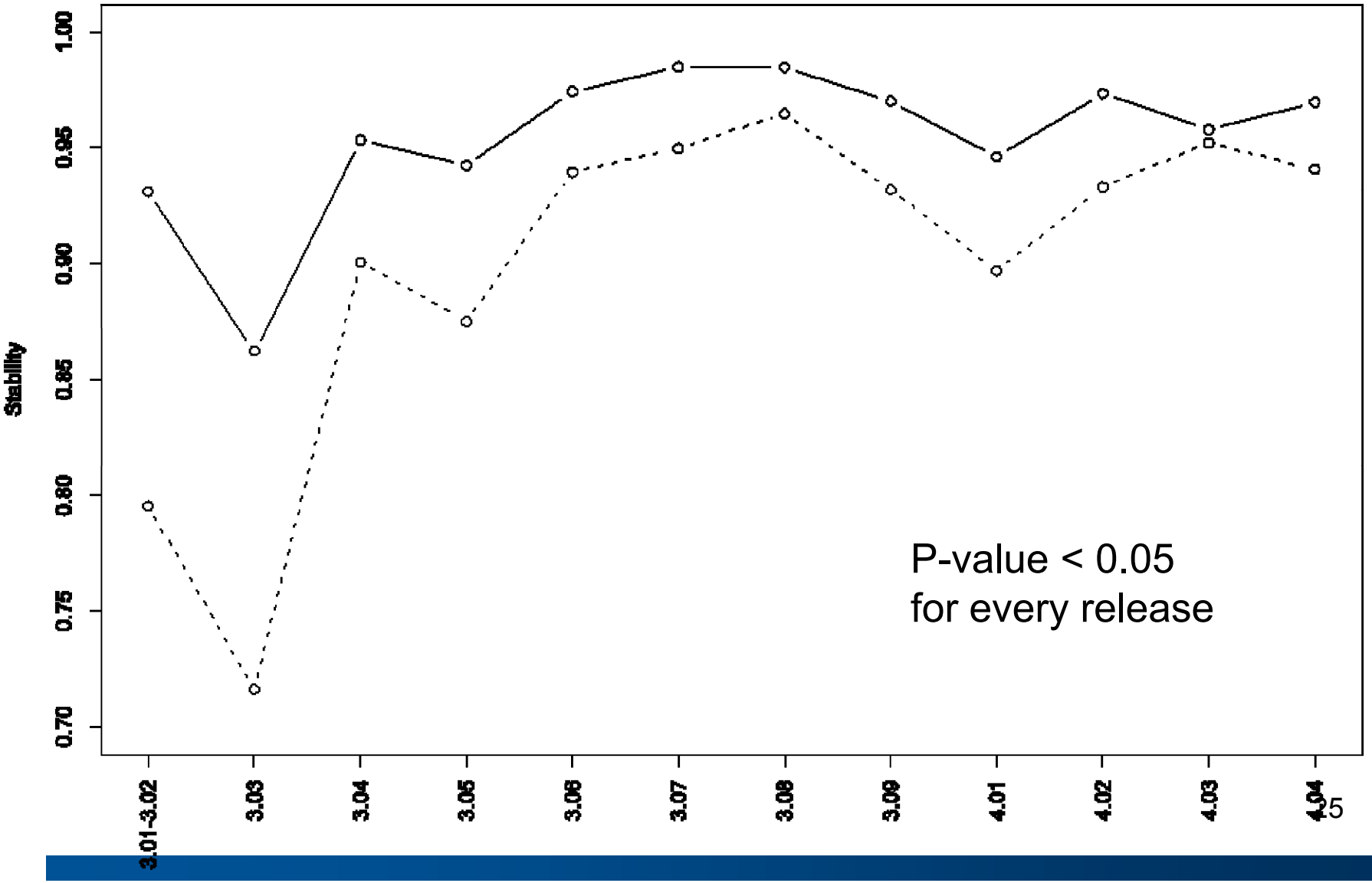


FONDAZIONE BRUNO KESSLER

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: Alice



FONDAZIONE
BRUNO KESSLER

Renaming

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

FONDAZIONE BRUNO KESSLER

Results

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.*

FONDAZIONE BRUNO KESSLER

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 (possibly 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.

FONDAZIONE BRUNO KESSLER

Empirical Study #2

RQ1: *How does the size of the source code vocabulary evolve over time?*

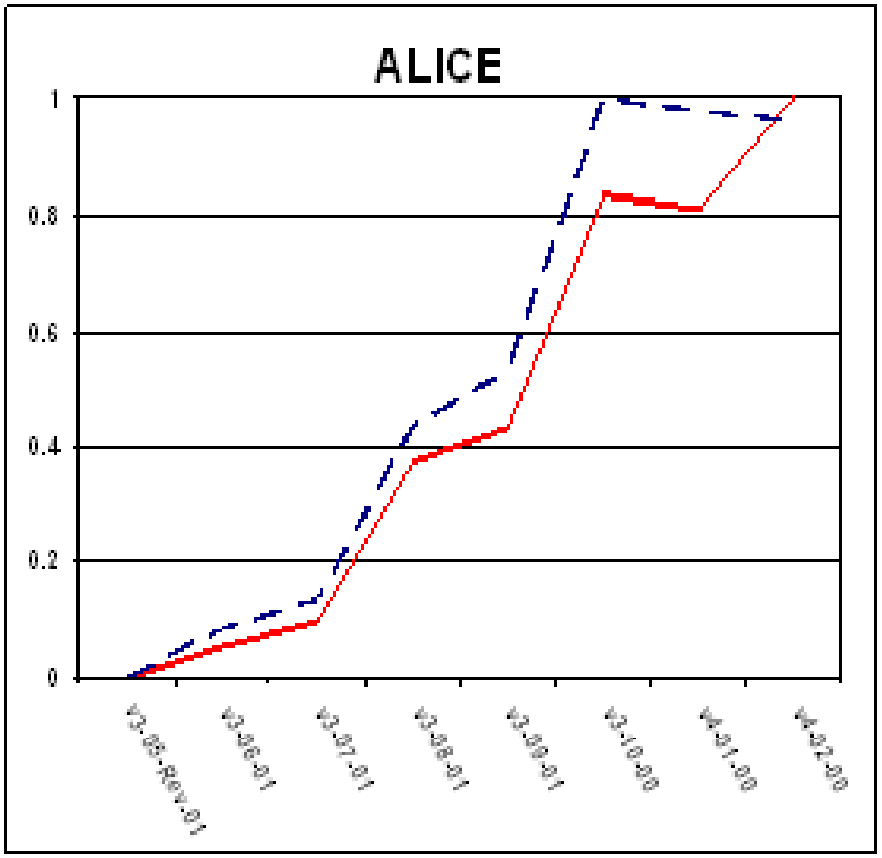
RQ2: *What are the relationships between individual vocabularies that form the source code vocabulary?*

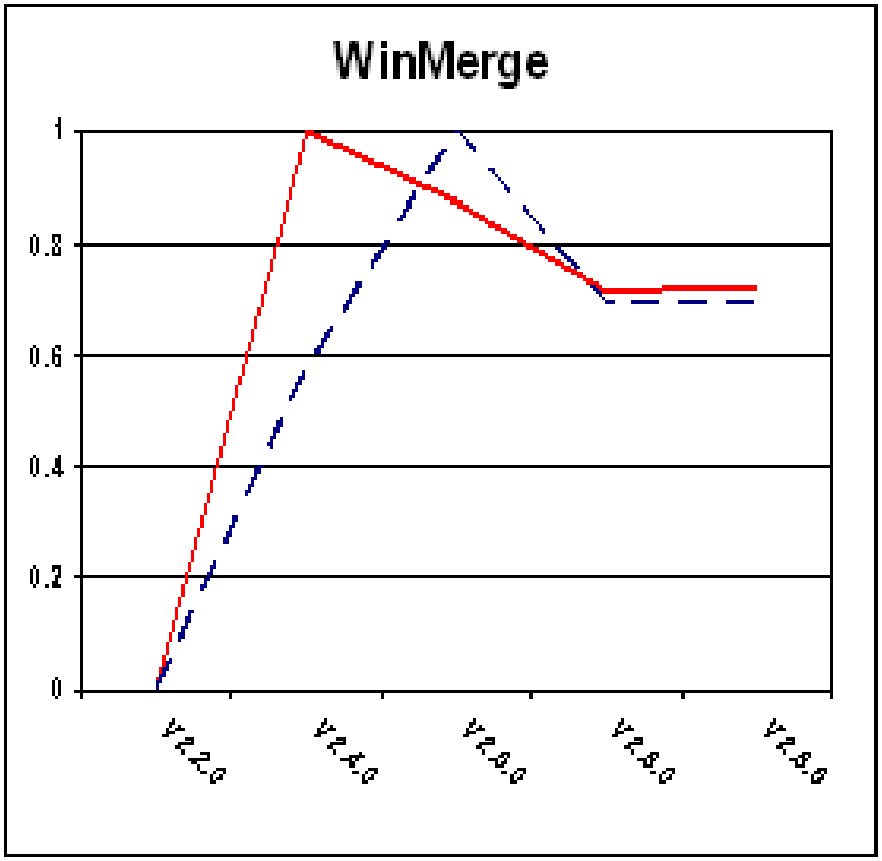
RQ3: *Are new identifiers introducing new terms?*

RQ4: *What do the most frequent terms refer to?*

FONDAZIONE
BRUNO KESSLER

Vocabulary size

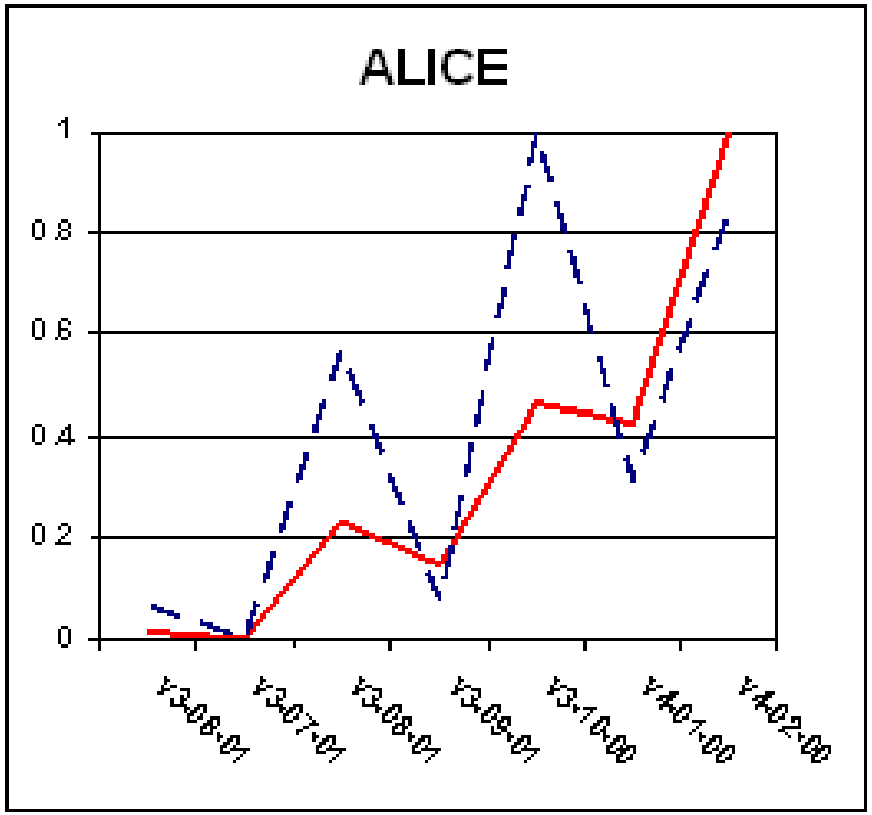


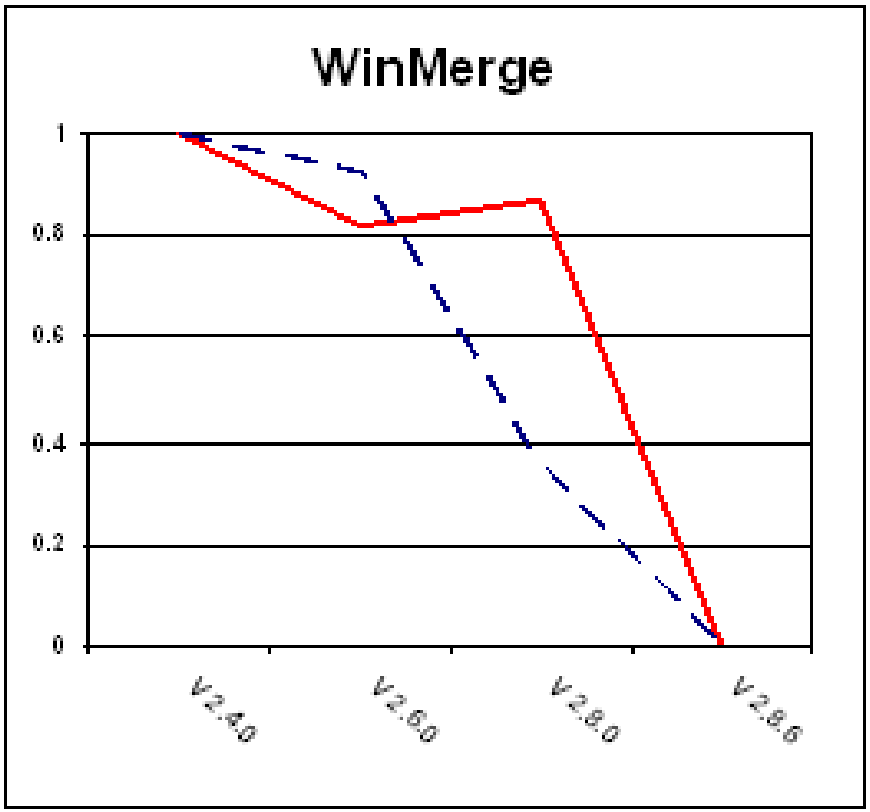


Vocabulary size (dashed) vs. code size (solid).

FONDAZIONE
BRUNO KESSLER

New terms

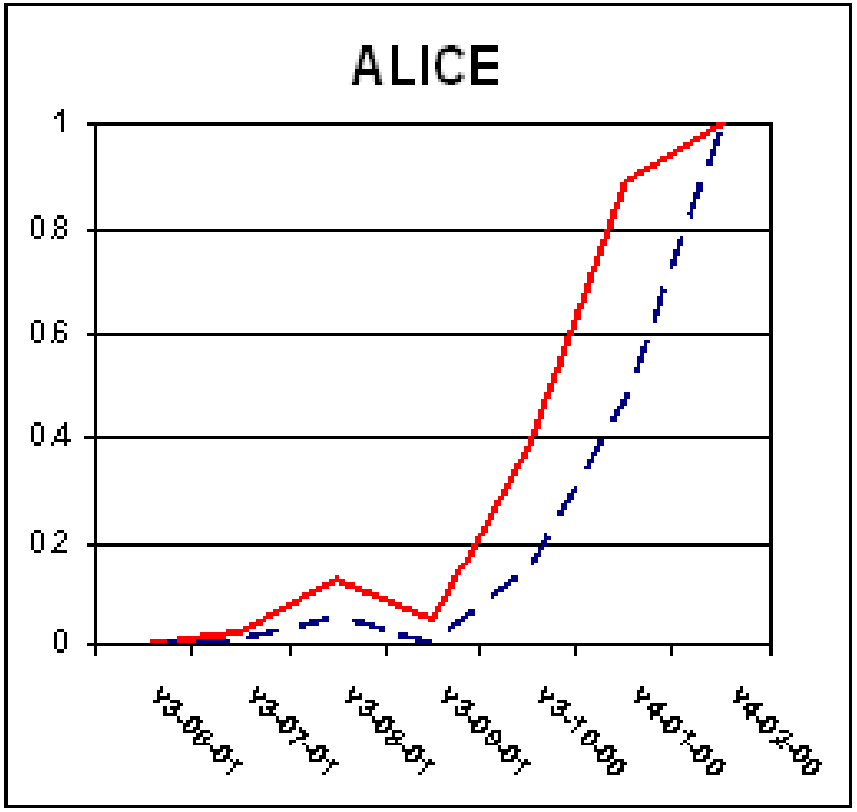


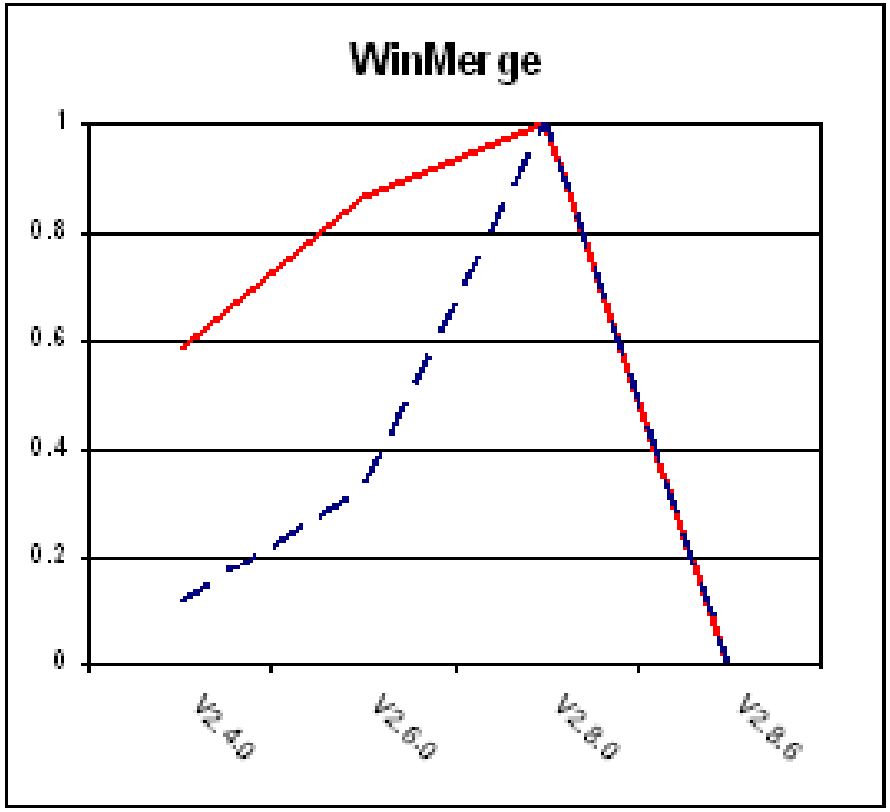


New terms (dashed) vs. new lines of code (solid).

FONDAZIONE
BRUNO KESSLER

Deleted terms





Deleted terms (dashed) vs. deleted lines of code (solid).

Commonalities across vocabularies

		ALICE	WinMerge			ALICE	WinMerge
CV∩AV	Size	126	147	**AV∩PV**	Size	510	371
	% from CV	59%	76%		% from AV	43%	60%
	% from AV	11%	24%		% from PV	48%	48%
CV∩FV	Size	211	191	**AV∩CoV**	Size	878	555
	% from CV	99.7%	98%		% from AV	75%	90%
	% from FV	16%	17%		% from CoV	13%	10%
CV∩PV	Size	112	124	**FV∩PV**	Size	492	443
	% from CV	53%	64%		% from FV	38%	58%
	% from PV	10%	16%		% from PV	46%	40%
CV∩CoV	Size	203	194	**FV∩CoV**	Size	1,076	955
	% from CV	96%	100%		% from FV	83%	85%
	% from CoV	3%	3%		% from CoV	15%	17%
AV∩FV	Size	749	466	**PV∩CoV**	Size	757	627
	% from AV	64%	75%		% from PV	71%	82%
	% from FV	58%	42%		% from CoV	11%	11%

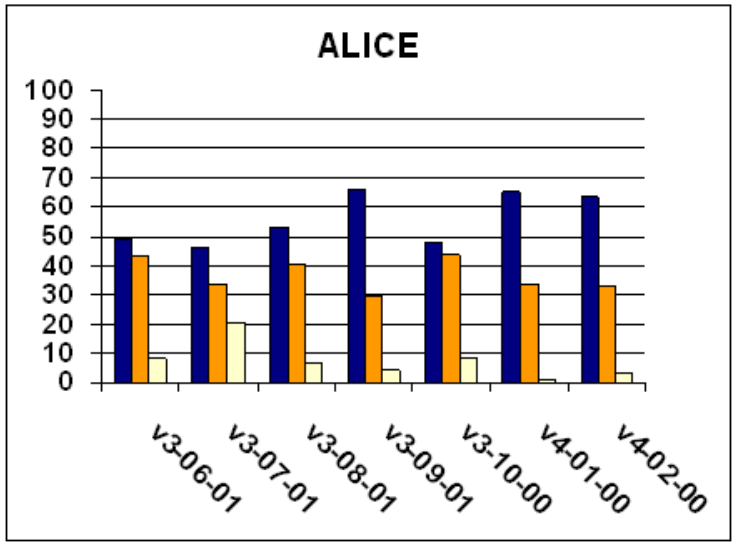
FONDAZIONE BRUNO KESSLER

New identifier terms appearing in comments

	ALICE	WinMerge
New CV∩CoV	30%	10%
New FV∩CoV	19%	16%
New AV∩CoV	15%	5%
New PV∩CoV	20%	14%

FONDAZIONE
BRUNO KESSLER

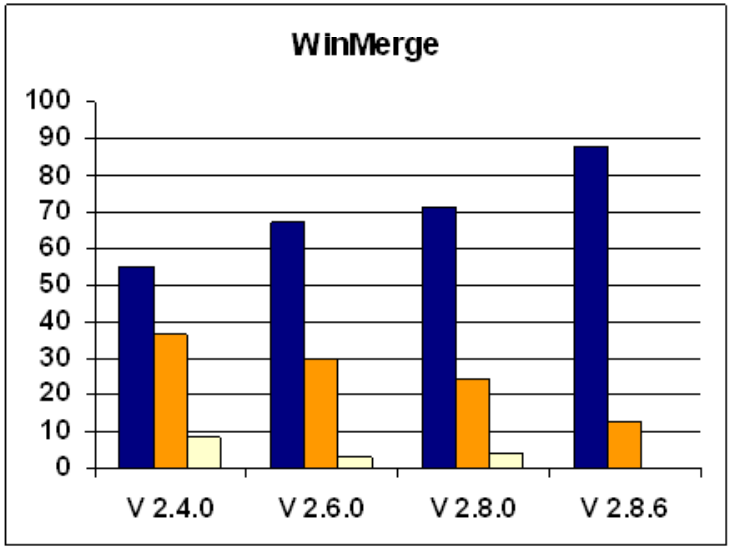
New identifiers



The % of new identifiers containing 0 new identifier terms

The % of new identifiers containing 1 new identifier terms

The % of new identifiers containing 2 or more new identifier terms



FONDAZIONE
BRUNO KESSLER

Most frequent terms

***ALICE**:*
hit, track, pad, digit, cluster, chamber, pho, muon, segment, rec, event, tr, rich, tpc, trigger, tof, mp, tp, phosrp, cpv, emc, ppsd, energy, ev, sector, rh.

FONDAZIONE
BRUNO KESSLER

Results

RQ1: System vocabulary and system size often exhibit a parallel evolution trend.

RQ2: The vocabulary used to build class identifiers has the largest number of terms in common with other vocabularies; identifier vocabulary changes are only marginally reflected in the comment vocabulary.

RQ3: New identifiers usually introduce no or at most one new term.

RQ4: Frequent terms are associated with core domain concepts.

FONDAZIONE BRUNO KESSLER

Discussion

- The vocabulary used by programmers is subject to an evolution pressure similar to that affecting code evolution.
- Keeping vocabulary evolution under control is a major challenge
- The quality of the vocabulary affects
  - how identifiers are constructed,
  - how core concepts are recognized and named, and
  - how difficult it is for a programmer to understand and evolve the system

From our study, we can envision a central role of class vocabulary terms and of frequent terms in recovering and structuring the knowledge conveyed by identifiers.

Improvement of the internal code quality

- Deal with natural language used in the code.
- Extract information about the natural language in the code and its evolution over time.
- Support natural language embedding and improvement by means of dedicated tools.

FONDAZIONE
BRUNO KESSLER

Research agenda

- Domain modeling through ontology and relationship between domain terms and identifiers used in programs.
- Tools for identifier construction and improvement, based on domain models: support to concise and consistent naming [Deissenboeck & Pizka, IWPC 2005]
- Tools for code understanding, searching and browsing: dictionary/glossary of terms used in a program; concept based search and code navigation.