Context-Aware Source Code Vocabulary Normalization for Software Maintenance

Presentation of the Ph.D. Defense

August 19, 2013

DGIGL - SOCCER Lab, Ptidej Team

École Polytechnique de Montréal, Québec, Canada

Latifa GUERROUJ

latifa.guerrouj@polymtl.ca

Outline

```java
package com.ibatis.common.beans;

import java.lang.reflect.*;
import java.math.*;
import java.util.*;

/**
 * This class represents a cached set of class definition information that
 * allows for easy mapping between property names and getter/setter methods.
 */
public class ClassInfo {

  private static boolean cacheEnabled = true;
  private static final String[] EMPTY_STRING_ARRAY = new String[0];
  private static final Set SIMPLE_TYPE_SET = new HashSet();
  private static final Map CLASS_INFO_MAP = Collections.synchronizedMap(new HashMap());

  private String className;
  private String[] readablePropertyNames = EMPTY_STRING_ARRAY;
  private String[] writeablePropertyNames = EMPTY_STRING_ARRAY;
  private HashMap setMethods = new HashMap();
  private HashMap getMethods = new HashMap();
  private HashMap setTypes = new HashMap();
  private HashMap getTypes = new HashMap();
  private Constructor defaultConstructor;

  static {
    SIMPLE_TYPE_SET.add(String.class);
    SIMPLE_TYPE_SET.add(Byte.class);
    SIMPLE_TYPE_SET.add(Short.class);
    SIMPLE_TYPE_SET.add(Character.class);
    SIMPLE_TYPE_SET.add(Integer.class);
    SIMPLE_TYPE_SET.add(Long.class);
    SIMPLE_TYPE_SET.add(Float.class);
```

Textual information embeds domain knowledge

* Deissenboeck, F. and Pizka , M., "Concise and Consistent Naming", Software Quality Journal, vol. 14, no. 3, 2006, pp. 261-282

```java
package com.ibatis.common.beans;

import java.lang.reflect.*;
import java.math.*;
import java.util.*;

/**
 * This class represents a cached set of class definition information that
 * allows for easy mapping between property names and getter/setter methods.
 */
public class ClassInfo {

  private static boolean cacheEnabled = true;
  private static final String[] EMPTY_STRING_ARRAY = new String[0];
  private static final Set SIMPLE_TYPE_SET = new HashSet();
  private static final Map CLASS_INFO_MAP = Collections.synchronizedMap

  private String className;
  private String[] readablePropertyNames = EMPTY_STRING_ARRAY;
  private String[] writeablePropertyNames = EMPTY_STRING_ARRAY;
  private HashMap setMethods = new HashMap();
  private HashMap getMethods = new HashMap();
  private HashMap setTypes = new HashMap();
  private HashMap getTypes = new HashMap();
  private Constructor defaultConstructor;

  static {
    SIMPLE_TYPE_SET.add(String.class);
    SIMPLE_TYPE_SET.add(Byte.class);
    SIMPLE_TYPE_SET.add(Short.class);
    SIMPLE_TYPE_SET.add(Character.class);
    SIMPLE_TYPE_SET.add(Integer.class);
    SIMPLE_TYPE_SET.add(Long.class);
    SIMPLE_TYPE_SET.add(Float.class);
```

Textual information embeds domain knowledge

About 70% of source code consists of identifiers*

Identifiers are important source of information for maintenance tasks such as:

- Traceability link recovery
- Feature location

* Deissenboeck, F. and Pizka , M., "Concise and Consistent Naming", Software Quality Journal, vol. 14, no. 3, 2006, pp. 261-282

```java
/**
 * Converts a DOM node to a complete xml string
 * @param node - the node to process
 * @param indent - how to indent the children of the node
 * @return The node as a String
 */
public static String nodeToString(Node node, String indent) {
  StringWriter stringWriter = new StringWriter();
  PrintWriter printWriter = new PrintWriter(stringWriter);

  switch (node.getNodeType()) {

    case Node.DOCUMENT_NODE:
      printWriter.println("<xml version=\"1.0\">\n");
      // recurse on each child
      NodeList nodes = node.getChildNodes();
      if (nodes != null) {
        for (int i = 0; i < nodes.getLength(); i++) {
          printWriter.print(nodeToString(nodes.item(i), ""));
        }
      }
      break;
```

Example of Java code using meaningful identifiers - ibatis

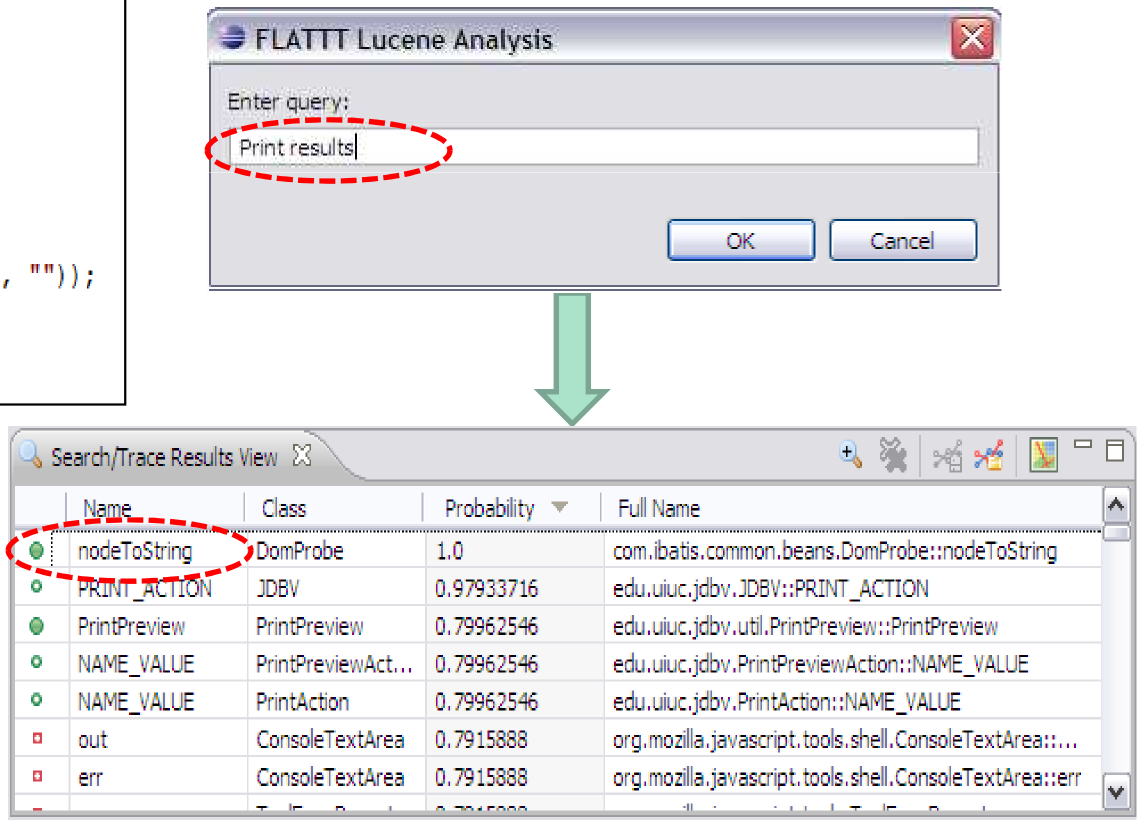
Enslen et al. (MSR'09):

Samurai: splits identifiers by mining terms frequencies in a large corpus of programs.

Lawrie et al. (WCRE'10, ICSM'11):

GenTest : generates all splittings and evaluates a scoring function against each one.

Nomalize: a refinement of GenTest towards expansion based on a machine-translation technique.



Example of Feature Location results - ibatis

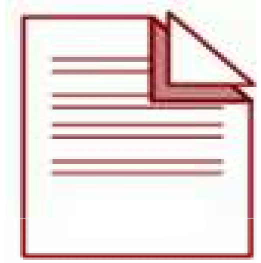
Research Context & Problem Statement

```
/* Size symbol. */
syms[2].the_bfd = abfd;
syms[2].name = mangle_name (abfd, "size");
syms[2].value = sec->_raw_size;
syms[2].flags = BSF_GLOBAL;
syms[2].section = bfd_abs_section_ptr;
syms[2].udata.p = NULL;

for (i = 0; i < BIN_SYMS; i++)
   *alocation++ = syms++;
*alocation = NULL;

return BIN_SYMS;
}
```

Vocabulary mismatch



Requirements

Example of C code identifiers - (gcl-2.6.7)

Normalizing Source Code Vocabulary !?

Normalization:

- **Splitting:** bfd abs section ptr

- **Expansion:** binary file descriptor absolute section pointer

Thesis

Overarching Research Question of the Thesis

Can we automatically resolve the vocabulary mismatch between source code and other software artifacts, using context, to support software maintenance tasks such as feature location and traceability recovery?

Thesis

Thesis Phases

- Context-Awareness for Source Code Vocabulary Normalization
  - **Context is relevant**
    (EMSE'13)
- Context-Aware Normalization Approaches (TIDIER & TRIS)
  - **TIDIER: Inspired by Speech Recognition**
    (CSMR10, JSEP'13)
  - **TRIS: Fast Solution Dealing with normalization as an Optimization Problem**
    (WCRE'12)
- Impact of Advanced Identifier Splitting on Traceability Recovery
  - **Advanced Identifier Splitting Can Help Traceability Recovery**
- Impact of Advanced Identifier Splitting on Feature Location
  - **Advanced Identifier Splitting Can Help Feature Location**
    (ICPC'11)

Contribution 1:

Context-Awareness for Source Code Vocabulary Normalization

Context-Awareness for Normalization

Experiments' Definition and Planning

Two experiments (Exp I and II) with 63 participants asked to split/expand identifiers from C programs with different contexts to investigate:

- Effect of contextual information;
- Accuracy in dealing with identifiers' terms consisting of plain English words, abbreviations, and acronyms;
- Effect of factors: participants' background, programming expertise, domain knowledge, and English proficiency.

Context-Awareness for Normalization

Exp I & II Subjects			
Characteristic	**Level**	**# of participants Exp I (42)**	**# of participants Exp II (21)**
Program of studies	Bachelor	5	3
	Master	9	6
	Ph.D.	28	10
	Post-doc	1	2
C Programming Experience	Basic	11	6
	Medium	23	5
	Expert	9	10
English Proficiency	Bad	8	1
	Good	8	9
	Very good	18	6
	Excellent	8 (7)	11(6)
Linux Knowledge	Occasional	12	10
	Basic usage	13	6
	Knowledgeable but not expert	17	5
	Expert	0	0

Participants' characteristics and background (63 participants in total).

Context-Awareness for Normalization

Objects: identifiers from # open-source C applications &…

GNU Projects (337 Projects)			
	C	**C++**	**.h**
Files	57, 268	13,445	39,257
Size (KLOCs)	25,442	2,846	6,062
Identifiers	1,154,280	-	619,652
Oracle	927	-	26

FreeBSD			
	C	**C++**	**.h**
Files	13,726	128	7,846
Size (KLOCs)	1,800	128	8,016
Identifiers	634,902	-	278,659
Oracle	20	-	0

Linux Kernel			
	C	**C++**	**.h**
Files	12,581	-	11,166
Size (KLOCs)	8,474	-	1,994
Identifiers	845,335	-	352,850
Oracle	73	-	4

Apache Web Server			
	C	**C++**	**.h**
Files	559	-	254
Size (KLOCs)	293	-	44
Identifiers	33,062	-	11,549
Oracle	11	-	0

Main characteristics of the 340 projects for the sampled identifiers.

Context-Awareness for Normalization

Context (Internal & External) made available to participants.

Context Levels	Exp I	Exp II
no context (control group)	✓	✓
function	✓	
file	✓	✓
file plus AF	✓	✓
application		✓
application plus AF		✓

Context levels provided during Exp I and Exp II (AF = Acronym Finder).

Experimental Design: Randomized Block Procedure

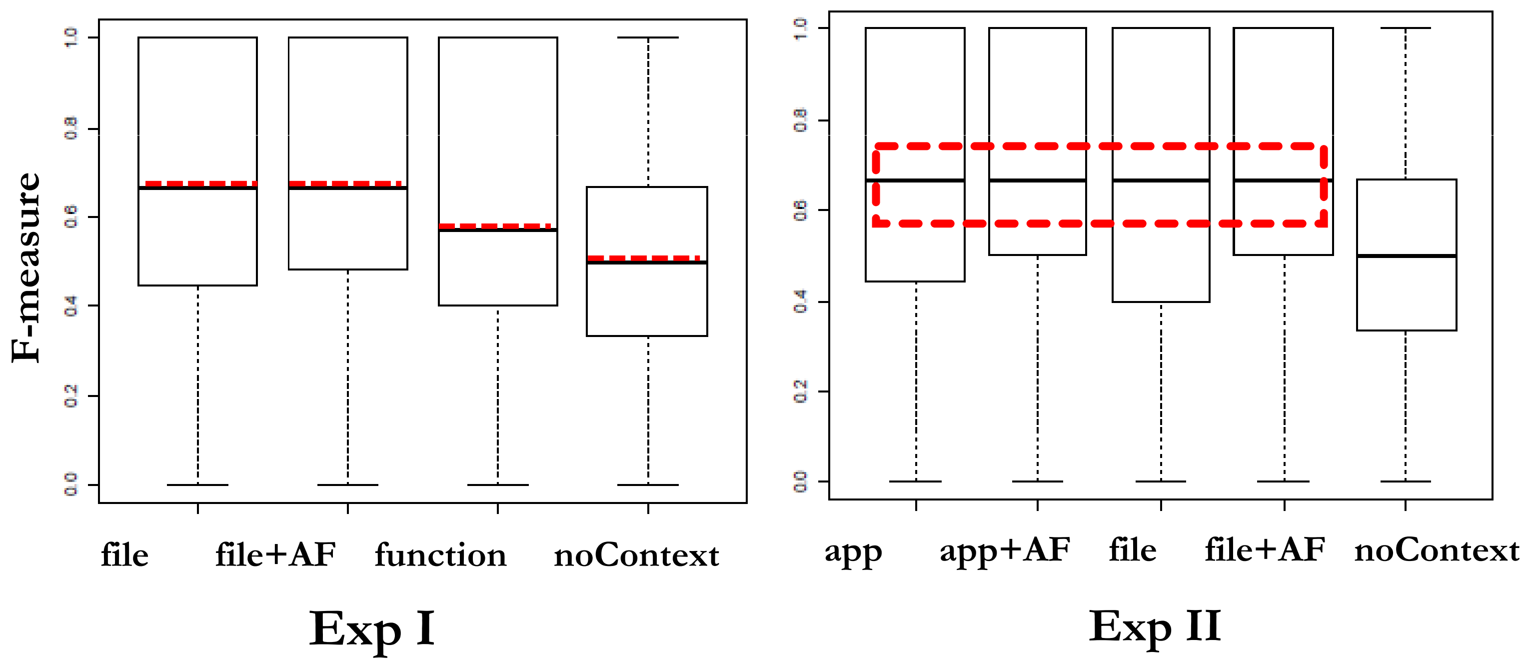
Context-Awareness for Normalization

Research Questions

- **RQ1**: To what extent does context impact splitting/expansion of identifiers?
- **RQ2**: To what extent do the characteristics of identifiers' terms affect the normalization performances?
- **RQ3**: To what extent do level of experience, programming language (C), domain knowledge, and English proficiency impact the normalization.

Context-Awareness for Normalization

Experiments' Results – RQ1 (Context Relevance)

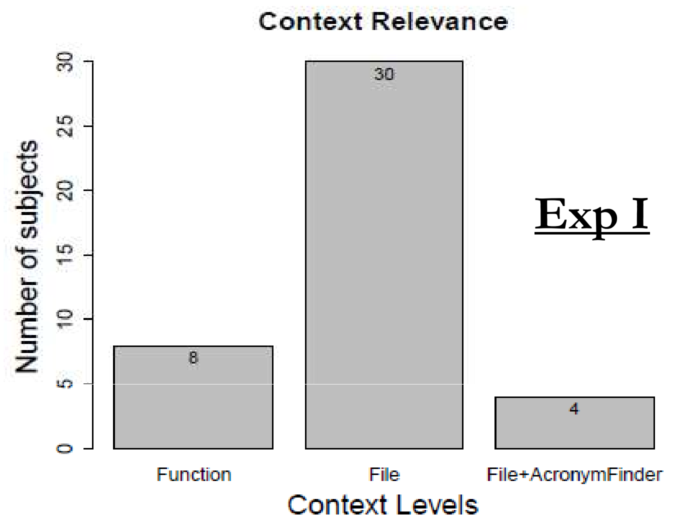


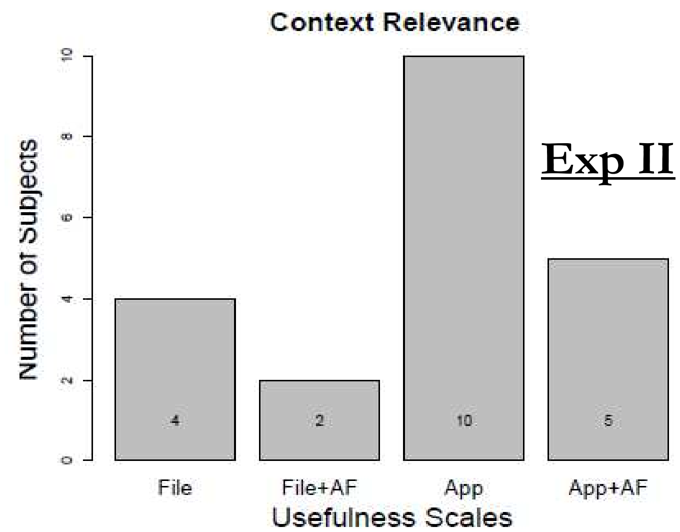
Boxplots of F-measure: Exp I and II context levels.

Context-Awareness for Normalization

Experiments' Results – RQ1 (Context Relevance)

- Context significantly increases participants' performances.
- File level exhibits better performances than the function-level context.
- Application-level context does not improve further.





Context-Awareness for Normalization

Experiments' Results – RQ2 (Effect of Kind of Terms)

Exp I				
Context	**Kind of Terms**	**#Matched**	**#Unmatched**	**Accuracy (%)**
file plus AF	abbreviation acronyms plain	523 112 336	169 31 50	**75.58** **78.32** **87.05**
file	abbreviation acronyms plain	542 94 346	164 32 50	**76.77** **74.60** **87.37**
function	abbreviation acronyms plain	582 97 374	161 36 52	**78.33** **72.93** **87.79**
no context	abbreviation acronyms plain	467 82 326	248 47 75	**65.31** **63.57** **81.30**
OVERALL	abbreviation acronym plain	2114 385 1382	742 146 227	**74.02** **72.50** **85.89**

Exp I: Proportions of kind of identifiers' terms correctly expanded per context level.

Context-Awareness for Normalization

Experiments' Results – RQ2 (Effect of Kind of Terms)

Exp II				
Context	**Kind of Terms**	**#Matched**	**#Unmatched**	**Accuracy (%)**
application plus AF	abbreviation	274	69	**79.88**
	acronyms	57	13	**81.43**
	plain	181	17	**91.41**
application	abbreviation	542	164	**75.35**
	acronyms	94	32	**82.61**
	plain	346	50	**90.45**
file plus AF	abbreviation	582	161	**82.87**
	acronyms	97	36	**86.30**
	plain	374	52	**91.67**
file	abbreviation	467	248	**76.60**
	acronyms	82	47	**85.07**
	plain	326	75	**92.57**
no context	abbreviation	2114	742	**67.98**
	acronym	385	146	**76.12**
	plain	1382	227	**83.94**
OVERALL	abbreviation	1349	415	**76.47**
	acronym	285	61	**82.37**
	plain	861	96	**89.97**

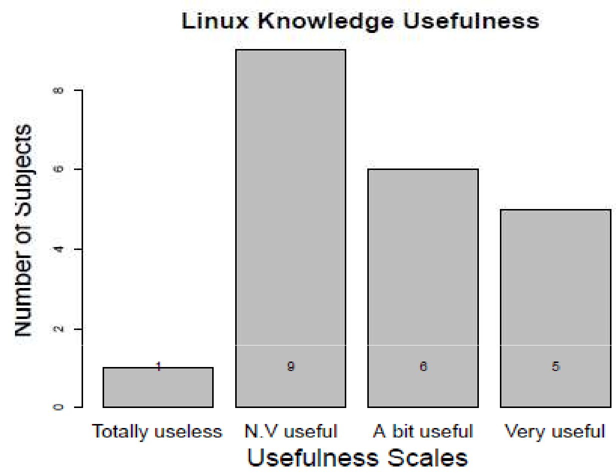
Exp II: Proportions of kind of identifiers' terms correctly expanded per context level.

Context-Awareness for Normalization

Experiments' Results – RQ3 (Effect of Part. Characteristics)

	Exp II
	p-value
Context	**<0.001**
Linux	**0.037**
Context:Linux	0.988

F-measure: two-way permutation test by context & knowledge of Linux.



Exp II

	Exp I	Exp II
	p-value	*p*-value
Context	**<0.001**	**<0.001**
English	**0.032**	**0.044**
Context:English	0.054	0.698

F-measure: two-way permutation test by context & English Proficiency.

Context-Awareness for Normalization

Conclusion

- Context is relevant for vocabulary normalization;
- No significant difference in the accuracy of splitting/expanding abbreviations and acronyms;
- Participants exploit better context when having a good level of English;
- English is used beside the domain knowledge (Exp II) to normalize identifiers.

Context is useful for source code vocabulary normalization

Contribution 2:

Context-Aware Source Code Vocabulary Normalization Approaches: **TIDIER** & TRIS

TIDIER Overview

Developers generate identifiers and contractions using:

- Terms and words reflecting domain concepts, developers' experience or knowledge;

- A finite set of transformation rules:

  - Dropping all vowels — pointer → pntr
  - Dropping a random vowel — user → usr
  - Dropping a random character — pntr → ptr
  - Dropping suffix (ing, tion, ment...) — available → avail
  - Dropping the last m characters — rectangle → rect

TIDIER Overview

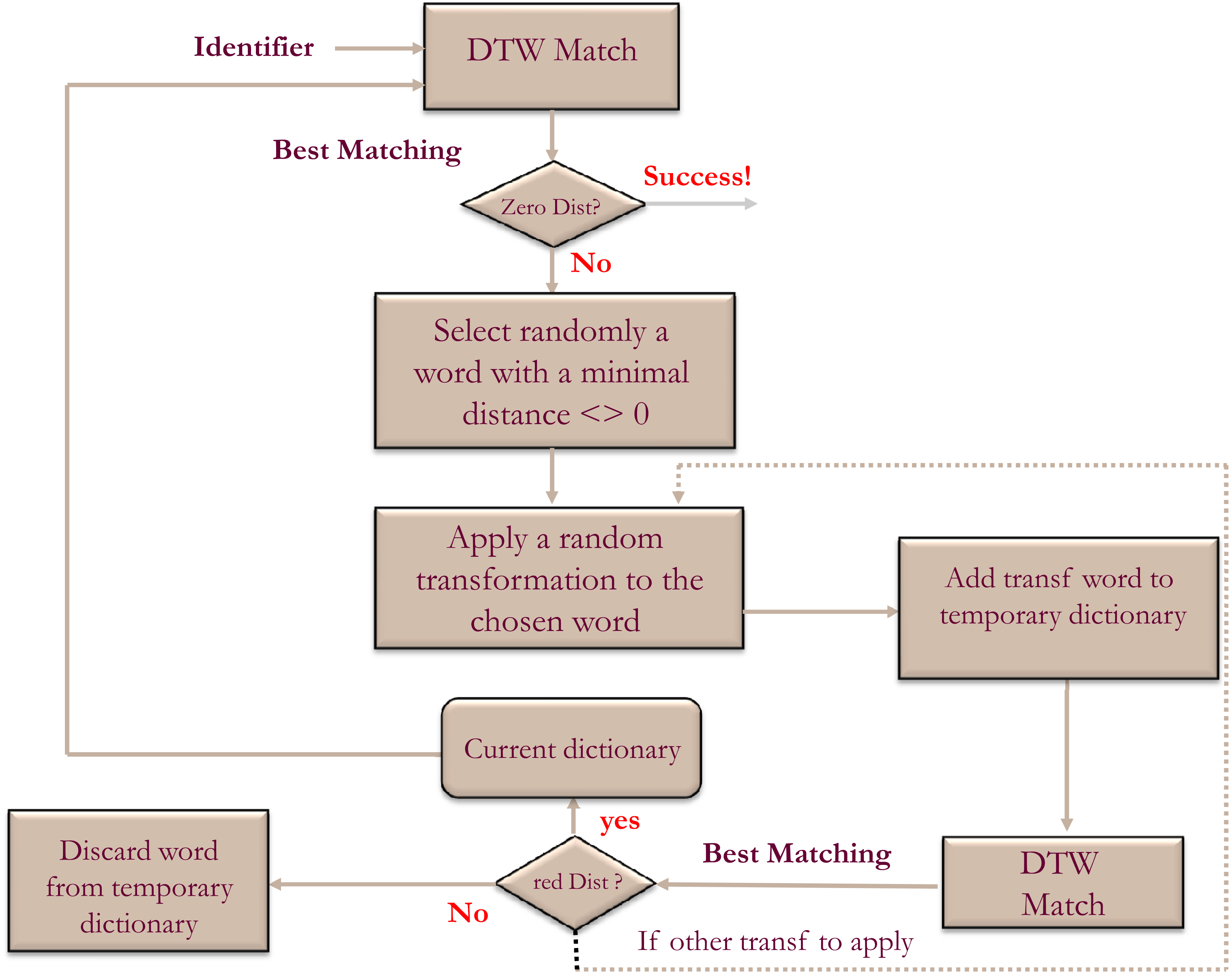
TIDIER is novel and uses context in the form of:

Context-aware dictionaries enriched by the use of domain knowledge.

TIDIER relies on a search-based technique to normalize identifiers:

- It relies on a distance using Dynamic Time Warping (DTW) for continuous speech recognition (Ney, IEE TSE'84);
- Hill Climbing.

TIDIER Normalization Strategy



TIDIER Case Study

Research Questions

- **RQ1**: How does TIDIER compare with alternatives when C identifiers must be split?
- **RQ2**: How sensitive are the performances of TIDIER to the use of context and specialized knowledge?
- **RQ3**: What percentage of identifiers with abbreviations is TIDIER able to map dictionary words?

Analyzed Systems (Benchmark used in Context study)

Camel Case & Samurai Techniques

Original Identifier	Camel Case
userId	user Id
setGID	set GID
print_file2device	print file 2 device
SSLCertificate	SSL Certificate
MINstring	MI Nstring
USERID	USERID
currentsize	currentsize
readadapterobject	readadapterobject
tolocale	tolocale
imitating	imitating
DEFMASKBit	DEFMASK Bit

Identifier Splitting for Traceability Recovery

Camel Case & Samurai Techniques

Original Identifier	Camel Case	Samurai
userId	user Id	user Id
setGID	set GID	set GID
print_file2device	print file 2 device	print file 2 device
SSLCertificate	SSL Certificate	SSL Certificate
MINstring	MI Nstring	MIN string
USERID	USERID	USER ID
currentsize	currentsize	current size
readadapterobject	readadapterobject	read adapter object
tolocale	tolocale	tol ocal e
imitating	imitating	imi ta ting
DEFMASKBit	DEFMASK Bit	DEF MASK Bit

Identifier Splitting for Traceability Recovery

Camel Case & Samurai Techniques

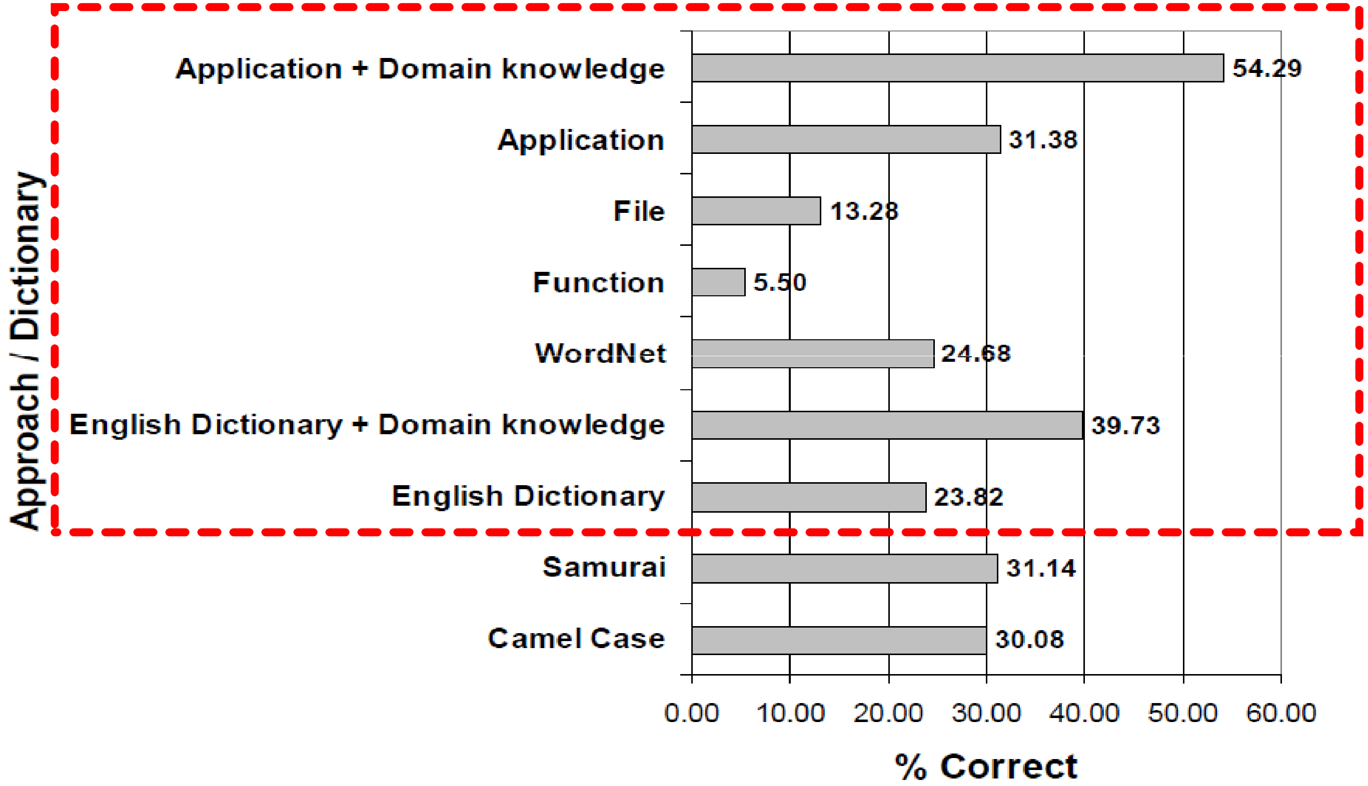
Original Identifier	Camel Case	Samurai
userId	user Id	user Id
setGID		set GID
print_file2device		print file 2 device
SSLCertificate	SSL Certi	SSL Certificate
MINstring	MI Nstring	MIN string
USERID	USERID	USER ID
currentsize	currentsize	current size
readadapterobject	readadapterobject	read adapter object
tolocale	tolocale	tol ocal e
imitating	imitating	imi ta ting
DEFMASKBit	DEFMASK Bit	DEF MASK Bit

Splits some cases where CamelCase cannot

Oversplits

TIDIER Results

Results



Performances of Camel Case, Samurai, and TIDIER when using different dictionaries.

TIDIER outperforms previous ones on C and it is the first to produce a correct mapping of 48% (35/73) for abbreviations.

Contribution 2:

Context-Aware Source Code Vocabulary Normalization Approaches: TIDIER & **TRIS**

TRIS Overview

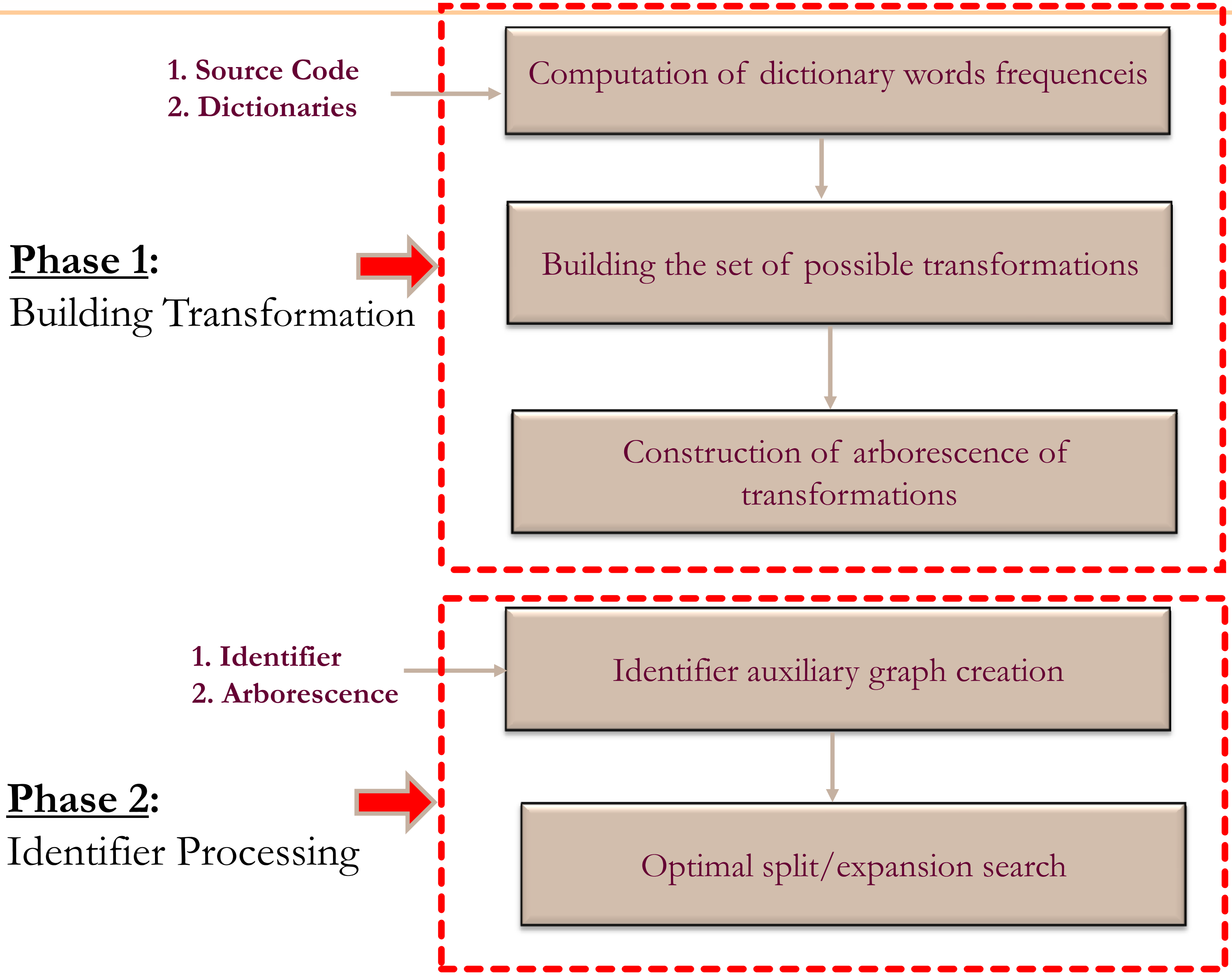
TRIS is a novel approach dealing with normalization as an **optimization (minimization) problem**:

The aim is to minimize the following cost function:

$$C(wOrig \rightarrow w) = \alpha * Freq(wOrig) + C(type(wOrig \rightarrow w))$$

- **Freq(wOrig)**: frequency of *wOrig* in the source code
- **C(type(wOrig→ w)**: cost of the transformation type

TRIS Normalization Strategy



TRIS Case Study

Research Question

RQ: What is the accuracy of the TRIS compared with alternative state-of-the art approaches?

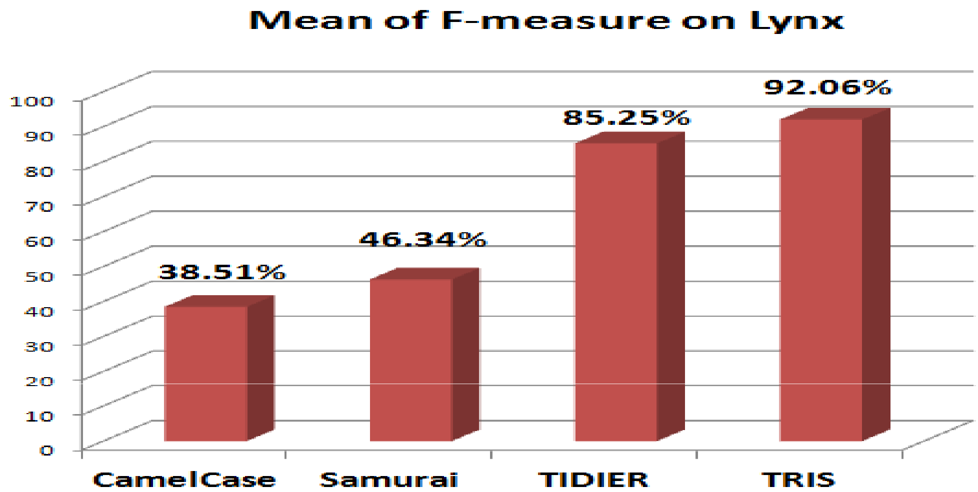
Analyzed Systems

JHotDraw – Java			
Files	Size (KLOC)	Identifiers	Oracle
155	16	2,348	957
Lynx - C			
Files	Size (KLOC)	Identifiers	Oracle
247	174	12,194	3,085
Lawrie et al. Data Set			
Programs	C (MLOC)	C++ (MLOC)	Java (MLOC)
186	26	15	7
489 C/C++ Sampled the Projects used in TIDIER			

Main characteristics of the systems analyzed using TRIS.

TRIS Results

Results



Mean of F-measure on Lynx (C system).

Approach 1	Approach 2	Adj *p*-value	Cliff's *d*
TRIS	Camel Case	**<0.001**	0.743
TRIS	Samurai	**<0.001**	0.684
TRIS	TIDIER	**<0.001**	0.204

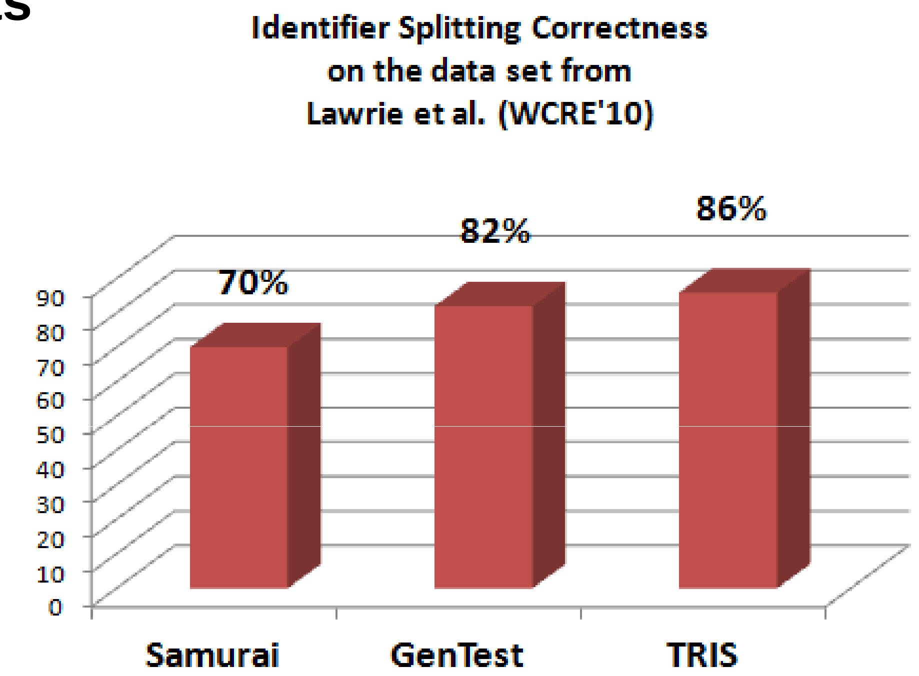
Results of Wilcoxon paired test & Cliff's Delta effect size on Lynx.

Cliff's delta Interpretation:

- **small**: 0.148 <= d <0.33, **medium**: 0.33 <= d < 0.474 and **large**: d >= 0.474

TRIS Results

Results



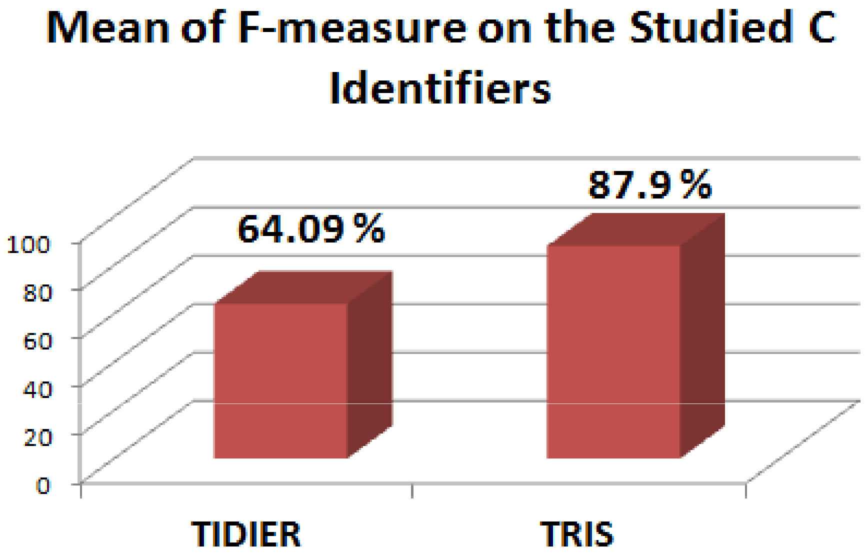
Identifier splitting correctness on the data set from *Lawrie et al.*

- **TRIS performs better than others with medium to large effect size on C;**

- **TRIS is better than Samurai of 16% and GenTest of 4%.**

TRIS Results

Results



Mean of F-measure on the 489 C sampled identifiers.

Statistically significant difference using Wilcoxon:

- **p-value < 0.001;**
- **Cliff's d effect size is medium (d = 0.456).**

Contribution 3:

Impact of Advanced Identifier Splitting on

Traceability Recovery

Identifier Splitting for Traceability Recovery

Research Question

RQ: How do different identifiers splitting strategies (CamelCase, Samurai and Oracle) impact Traceability Recovery?

Traceability Recovery Techniques Configurations

Splitting strategy	LSI	VSM
CamelCase	$LSI_{CamelCase}$	$VSM_{CamelCase}$
Samurai	$LSI_{Samurai}$	$VSM_{Samurai}$
Oracle	LSI_{Oracle}	VSM_{Oracle}

Configurations of the studied Traceability Recovery techniques.

Identifier Splitting for Traceability Recovery

Analyzed Systems

Systems (Java)	Version	# Requirements	# Classes
iTrust	10	35	218
Pooka	2.0	90	298

System (C)	Version	# Files	Size (KLOCs)	# Methods
Lynx	2.8.5	247	174	2,067

Main characteristics of the studied systems.

Identifier Splitting for Traceability Recovery

Results (%)

Systems	Precision			Recall		
	$LSI_{CamelCase}$	$LSI_{Samurai}$	LSI_{Oracle}	$LSI_{CamelCase}$	$LSI_{Samurai}$	LSI_{Oracle}
iTrust	36.49	36.49	28.39	36.61	36.61	34.23
Pooka	14.06	14.14	**15.64**	22.81	22.37	22.36
Lynx	45.43	39.08	39.40	41.99	40.82	41.55

Systems	Precision			Recall		
	$VSM_{CamelCase}$	$VSM_{Samurai}$	VSM_{Oracle}	$VSM_{CamelCase}$	$VSM_{Samurai}$	VSM_{Oracle}
iTrust	48.99	48.99	25.81	23.77	23.77	23.07
Pooka	40.54	40.54	**42.07**	11.59	11.63	**12.19**
Lynx	64.26	57.84	49.91	37.66	37.05	**40.16**

Precision and Recall of the Traceability Recovery techniques configurations for iTrust, Pooka, and Lynx.

Results and Discussion

- **Potential benefits of developing advanced vocabulary normalization approaches**.
- **Mismatch resulting from the requirements** (presence of acronyms in requirements).
- Case of Lynx (noise in data) : requirement 534 is "the browser should be able to manage store erase session I information". Whereas a C method *LYMain.c.i__nobrowse_fun* is related to browse directories functionality.
- Baseline splitting: "nobrowse" and thus no link between requirement 534 and *LYMain.c.i_nobrowse_fun.txt*.
- Samurai and manual oracle split the identifier "nobrowse" into "no browse" and link the file *LYMain.c.i__nobrowse_fun.txt*.

Potential benefits of developing advanced normalization approaches

Contribution 4:

Impact of Advanced Identifier Splitting on **Feature Location**

Identifier Splitting for Feature Location

Research Question

RQ: How do different identifiers splitting strategies (CamelCase, Samurai and Oracle) impact Feature Location?

Feature Location Techniques (FLTs) Configurations

Splitting strategy	IR FLT	IR_{Dyn} FLT
CamelCase	$IR_{CamelCase}$	$IR_{CamelCaseDyn}$
Samurai	$IR_{Samurai}$	$IR_{SamuraiDyn}$
Oracle	IR_{Oracle}	$IR_{OracleDyn}$

Feature Location techniques configurations studied.

Identifier Splitting for Feature Location

Analyzed Systems

System Version	Size (KLOC)	Classes	Methods	# Data Sets
Rhino 1.6R5	32	138	1,870	Eaddy et al.'s data* (2)
jEdit 4.3	109	483	6.4	2

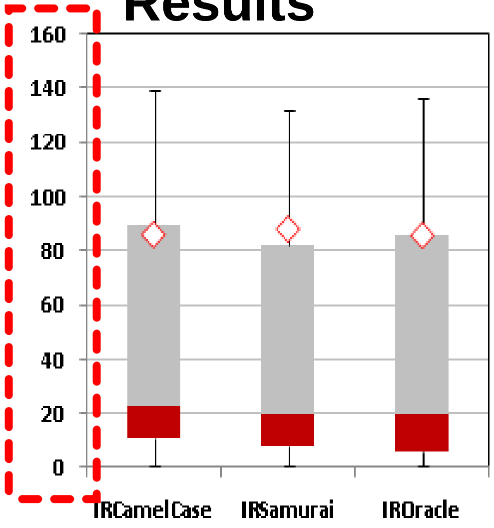
Dataset	Size	Queries	Gold Sets	Execution Information
$\text{Rhino}_{\text{Features}}$	241	Sections of ECMAScript	Eaddy et al.*	Full Execution Traces (from unit tests)
$\text{Rhino}_{\text{Bugs}}$	143	Bug title and description	Eaddy et al.* (CVS)	N/A
$\text{jEdit}_{\text{Features}}$	64	Feature (or Patch) title and description	SVN	Marked Execution Traces
$\text{jEdit}_{\text{Bugs}}$	86	Bug title and description	SVN	Marked Execution Traces

Characteristics of the main analyzed systems.

*** http://www.cs.columbia.edu/~eaddy/concerntagger/**

Identifier Splitting for Feature Location

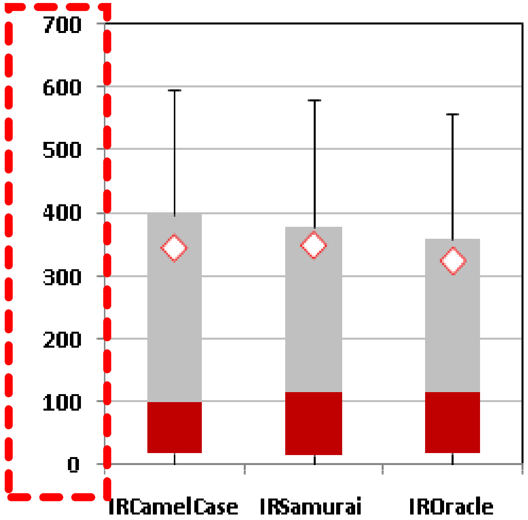
Results



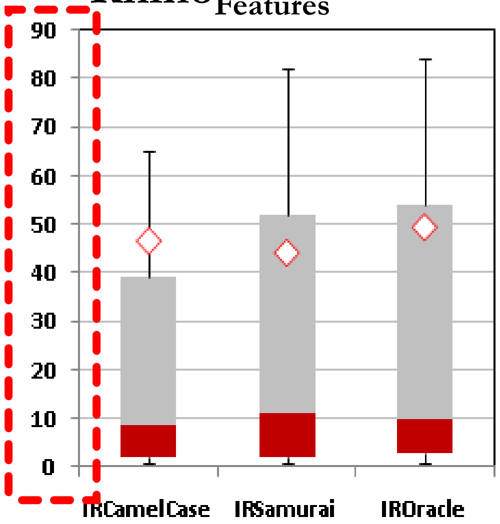
Rhino$_{Features}$

Similar median & average of **effectiveness measure**

IR FLTs

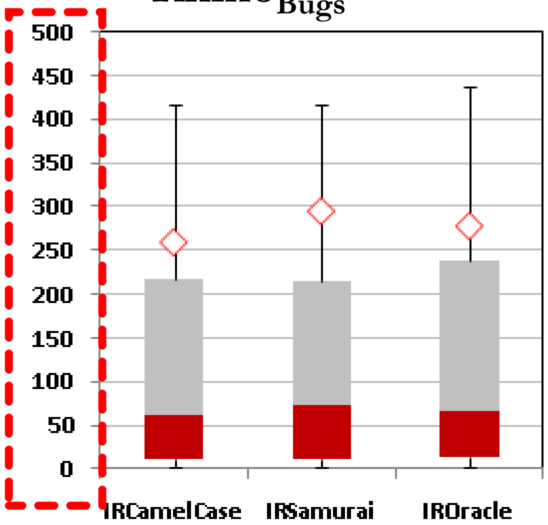


Rhino$_{Bugs}$



jEdit$_{Features}$

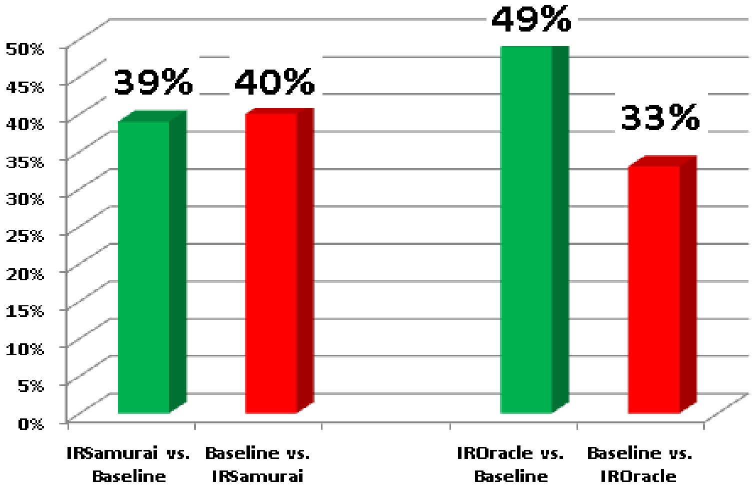
Datasets with features have better results than datasets with bugs



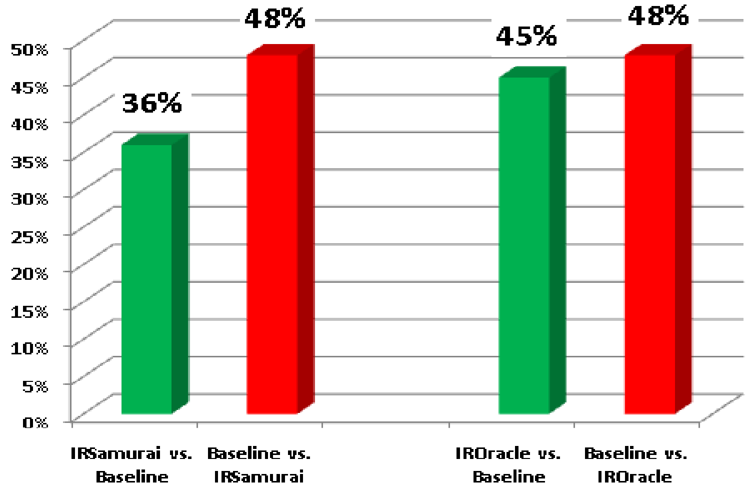
jEdit$_{Bugs}$

Identifier Splitting for Feature Location

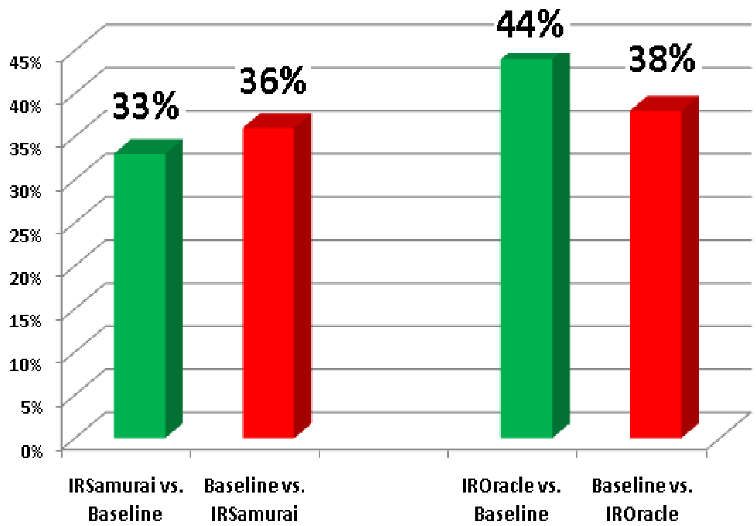
Results



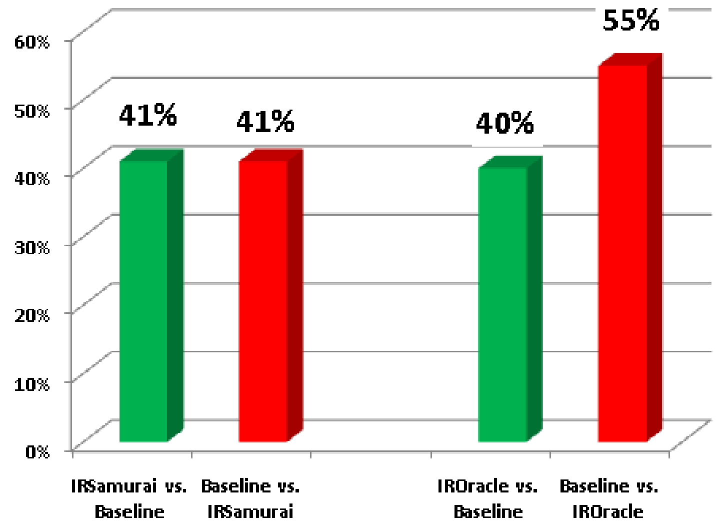
Rhino$_{Features}$



Rhino$_{Bugs}$



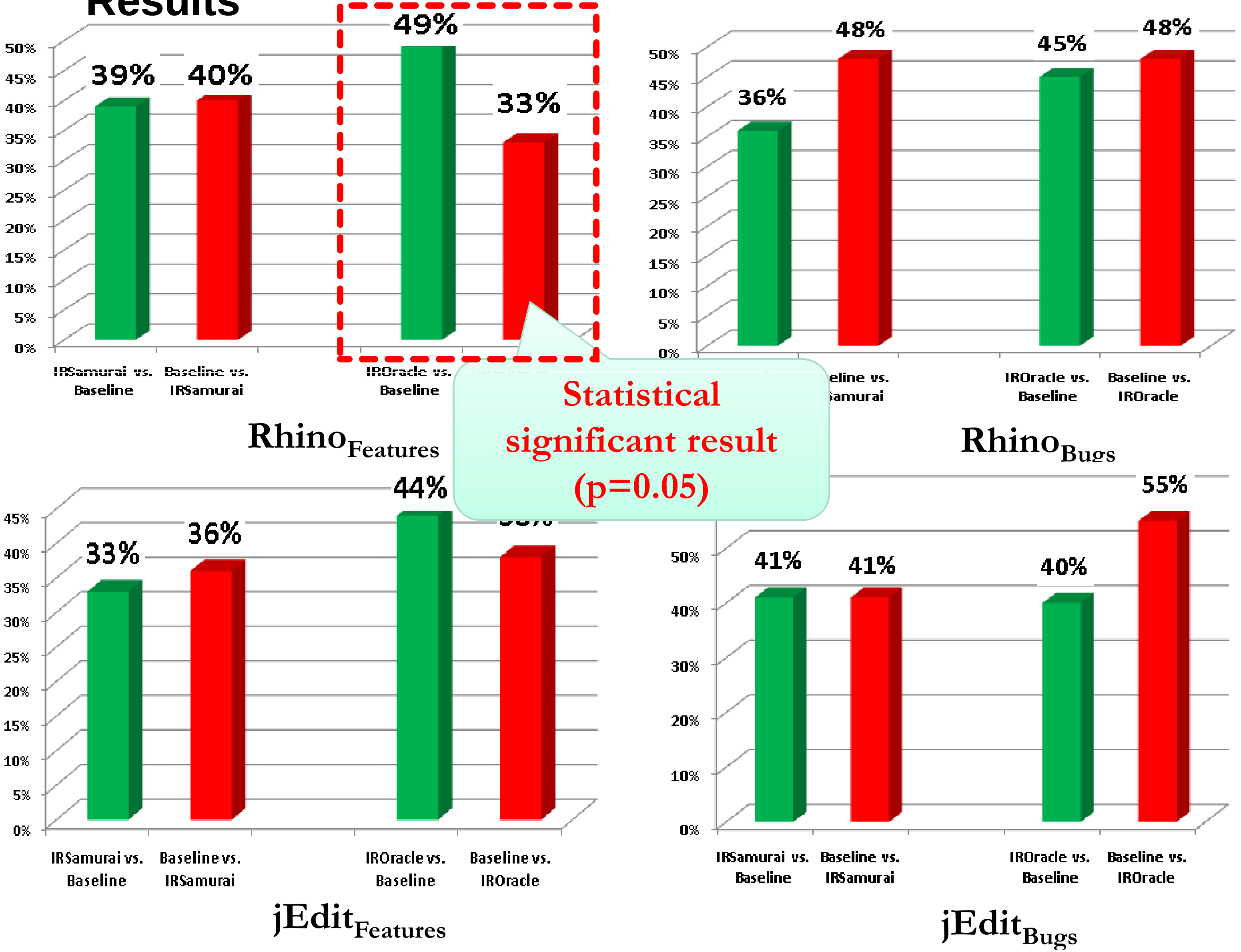
jEdit$_{Features}$



jEdit$_{Bugs}$

Identifier Splitting for Feature Location

Results



Identifier Splitting for Feature Location

Results and Discussion

- Samurai and CamelCase produced similar results;

- IR_{Oracle} outperforms $IR_{CamelCase}$ in terms of the effectiveness measure, on the $Rhino_{Features}$ dataset;

- When only textual information is available, an improved splitting technique can help improve effectiveness of feature location.

- **Samurai ovesplits identifiers into many meaningless** terms. In Rhino: *debugAccelerators* to debug *Ac ce le r at o rs* (CamelCase better in such cases).

Identifier Splitting for Feature Location

Vocabulary mismatch between queries and code

- Inconsistencies between the **identifiers used in the queries**, and the **identifiers used in the code**.

- The mismatch is less noticeable for features and more severe for bugs.

- **jEdit's feature #16084869** ("Support "thick" caret") contained in its description identifiers found in the name of the methods (e.g., thick, caret, text, area, etc.).

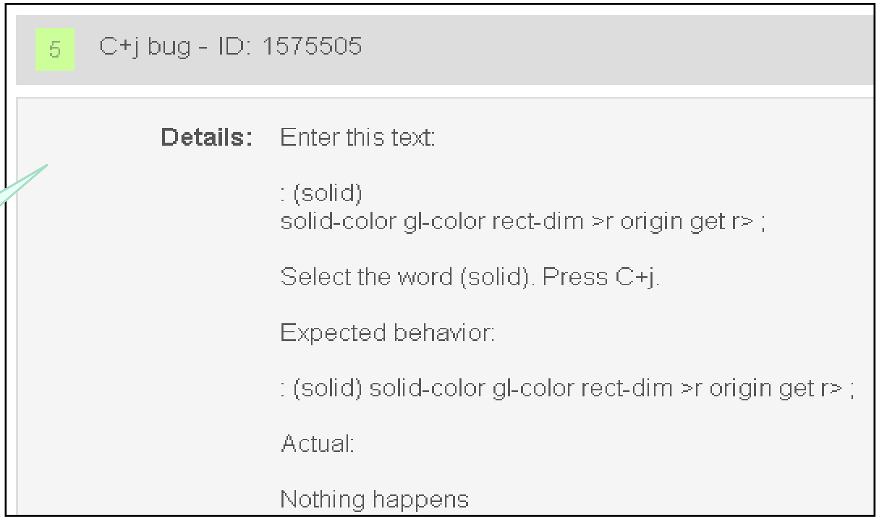
  - Name of developers (e.g., Slava,Carlos- Identifiers specific to communication (e.g., thanks, greetings, annoying).

  - Appeared only in the query vocabulary, and did not appear in the source code vocabulary.

Identifier Splitting for Feature Location

Features are more "descriptive" than bugs

Words "join" and "line" are not mentioned



Example of query (bugs)

Binkley et al. (ICSM'12): Normalization improves Feature Location

Potential benefits of developing advanced normalization approaches

Conclusion

Context is relevant for source code vocabulary normalization.
Source code files are the most helpful
A **limited context** such as **functions** does not help
A **wider context** such as **applications** does not improve further.

Domain knowledge improves normalization.

TIDIER is **novel** and **performs better** than its previous approaches (CamelCase & Samurai):
54.29% of splitting correctness vs. **31.14%** for **(Samurai)** & **30.08% (Camel Case)** with an application level dictionary augmented with domain knowledge
TIDIER was the **first** to produce a correct mapping for **48%** of abbreviations.

TRIS is novel and brings improvements on state-of-the-art approaches on C:
92.06% vs. **85.25%** for **TIDIER** (Lynx- C)
vs. **46.34%** for **Samurai**
vs. **38.51%** for **CamelCase**
86% vs. **82%** for **GenTest** on Lawrie et al. data
vs. **70%** for **Samurai**.
87.90% vs. **64.09%** for **TIDIER** on the identifiers from the 340 projects.

Advanced identifier splitting strategies improves the average of precision and recall of some systems: **Pooka & Lynx.**

Advanced splitting improves feature location using LSI: **Rhino (features).**

The **quality** of the **requirements** and expressiveness of the **queries impact** too.

Future Work

Impact of Vocabulary Normalization on Maintenance Tasks

- Evaluate our work on other systems such as C, C++ or COBOL;
- Compare it to other works such as Normalize (Lawrie et al, ICSM'11);
- Study the impact of IR queries quality (Haiduc et al. (ICSE'13)).

Context-Aware Vocabulary Normalization Approaches

- Extend the evaluation of TIDIER and TRIS on larger systems;
- Compare the results to more recent approaches such as Normalize (Lawrie et al., ICSM'11) and LINSEN (Corazza et al., ICSM'12).

Future Work

Context-Awareness for Vocabulary Normalization

- Replicate our studies using eye-tracking tools;
- Implement a context model that within an IDE support program understanding;
- Involve participants from industry.

Mining Software Repositories to Study the Impact of Identifier Style on Software Quality

- Infer the identifier styles in open-source projects using HMM;
- Analyze whether open-source developers adapt/bring their style;
- Analyze whether identifier style can introduce bugs and--or impacts internal quality metrics such as semantic coupling & cohesion.

Publications

Articles in journals

1. **Latifa Guerrouj**, Massimilano Di Penta, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. An Experimental Investigation on the Effects of Contexts on Source Code Identifiers Splitting and Expansion. Empirical Software Engineering Journal (EMSE'13).

2. **Latifa Guerrouj**, Massimilano Di Penta, Giuliano Antoniol, and Yann-Gaël Guéhéneuc. TIDIER: An Identifier Splitting Approach Using Speech Recognition Techniques. Journal of Software Evolution and Process (JSEP'13). 25(6): 569-661.

Conference Articles

3. **Latifa Guerrouj**, Philippe Galinier, Yann-Gaël Guéhéneuc, Giuliano Antoniol, and Massimiliano Di Penta. TRIS: a Fast and Accurate Identifiers Splitting and Expansion Algorithm. Proceedings of the 19th IEEE Working Conference on Reverse Engineering (WCRE), October 2012.

4. Bogdan Dit, **Latifa Guerrouj**, Denys Poshyvanyk, Giuliano Antoniol. Can Better Identifier Splitting Techniques Help Feature Location? Proceedings of the 19 IEEE International Conference on Program Comprehension (ICPC), June 2011.

Publications

Conference Articles

5. Nioosha Madani, **Latifa Guerrouj,** Massimiliano Di Penta, Yann-Gaël Guéhéneuc, Giuliano Antoniol. Recognizing Words from Source Code Identifiers Using Speech Recognition Techniques. Proceedings of the 14th IEEE European Conference on Software Maintenance and Reengineering (CSMR), Mars 2010. **Best Paper award of CSMR'10**.

6. **Latifa Guerrouj.** Normalizing Source Code Vocabulary to Enhance Program Comprehension and Software Quality. Proceedings of the 35th ACM International Conference on Software Engineering (ICSE), May 2013.

7. **Latifa Guerrouj.** Automatic Derivation of Concepts Based on the Analysis of Source Code Identifiers. Proceedings of the 17th Working Conference on Reverse Engineering (WCRE), October 2012.

8. Alberto Bacchelli, Nicolas Bettenburg, **Latifa Guerrouj.** Mining Unstructured Data because "Mining Unstructured Data is Like Fishing in Muddy Waters!". Proceedings of the 19th Working Conference on Reverse Engineering (WCRE), October 2012.

Conclusion

Context is relevant for source code vocabulary normalization.
Source code files are the most helpful
A **limited context** such as **functions** does not help
A **wider context** such as **applications** does not improve further.

Domain knowledge improves normalization.

TIDIER is **novel** and **performs better** than its previous approaches (CamelCase & Samurai):
54.29% of splitting correctness vs. **31.14%** for **(Samurai)** & **30.08% (Camel Case)** with an application level dictionary augmented with domain knowledge
TIDIER was the **first** to produce a correct mapping for **48%** of abbreviations.

TRIS is novel and brings improvements on state-of-the-art approaches on C:
92.06% vs. **85.25%** for **TIDIER** (Lynx- C)
vs. **46.34%** for **Samurai**
vs. **38.51%** for **CamelCase**
86% vs. **82%** for **GenTest** on Lawrie et al. data
vs. **70%** for **Samurai**.
87.90% vs. **64.09%** for **TIDIER** on the identifiers from the 340 projects.

Advanced identifier splitting strategies improves the average of precision and recall of some systems: **Pooka & Lynx.**

Advanced splitting improves feature location using LSI: **Rhino (features).**

The **quality** of the **requirements** and expressiveness of the **queries impact** too.

References

LAWRIE, D., FEILD, H. et BINKLEY, D. (2006). Syntactic Identifier Conciseness and Consistency. Proceedings of the 6th International Workshop on Source Code Analysis and Manipulation. pp. 139–148.

MAYRHAUSER, A. V. et VANS, A. M. (1995). Program Comprehension During Software Maintenance and Evolution. Computer, vol. 28, pp. 44–55

M-A.D. STOREY, F.D. FRACCHIA, H. M. (1999). Cognitive Design Elements to Support the Construction of a Mental Model During Software Exploration. Journal of Systems and Software, vol. 44, pp. 171–185.

ROBILLARD, M. P., COELHO, W. et MURPHY, G. C. (2004). How Effective Developers Investigate Source Code: An Exploratory Study. IEEE Transactions on Software Engineering, vol. 30, pp. 889–903.

KERSTEN, M. et MURPHY, G. C. (2006). Using Task Context to Improve Programmer Productivity. Proceedings of the 14th International Symposium on Foundations of Software Engineering. pp. 1–11.

SILLITO, J., MURPHY, G. C. et VOLDER, K. D. (2008). Asking and Answering Questions during a Programming Change Task. IEEE Transactions on Software Engineering, vol. 34,pp. 434–451.

BINKLEY, D., DAVIS, M., LAWRIE, D. et MORRELL, C. (2009). To Camelcase or Under score. Proceedings of the 17th International Conference on Program Comprehension. pp. 158–167.

ENSLEN, E., HILL, E., POLLOCK, L. et SHANKER, K. V. (2009). Mining Source Code to Automatically Split Identifiers for Software Analysis. Proceedings of the 6th International Working Conference on Mining Software Repositories. pp. 16–17.

LAWRIE, D. J., BINKLEY, D. et MORRELL, C. (2010). Normalizing Source Code Vocabulary. Proceedings of the 17th Working Conference on Reverse Engineering. pp. 112–122.

References

LAWRIE, D. et BINKLEY, D. (2011). Expanding Identifiers to Normalize Source Code Vocabulary. Proceedings of the 27th International Conference on Software Maintenance. pp. 113–122.

CORAZZA, A., MARTINO, S. D. et MAGGIO, V. (2012). LINSEN: An Efficient Approach to Split Identifiers and Expand Abbreviations. Proceedings of the 28th International Conference of Software Maintenance. pp. 233–242.

EISENBARTH, T., KOSCHKE, R. et SIMON, D. (2003). Locating Features in Source Code. IEEE Transactions on Software Engineering, vol. 29, pp. 210–224.

POSHYVANYK, D., GU´EH´ENEUC, Y.-G., MARCUS, A., ANTONIOL, G. et RAJLICH, V. (2007). Feature Location Using Probabilistic Ranking of Methods Based on Execution Scenarios and Information Retrieval. IEEE Transactions on Software Engineering, vol. 33, pp. 420–432.

EADDY, M., AHO, A., ANTONIOL, G. et GU´EH´ENEUC, Y.-G. (2008a). CERBERUS: Tracing Requirements to Source Code Using Information Retrieval, Dynamic Analysis, and Program Analysis. Proceedings of 16th International Conference on Program Comprehension. pp. 53–62.

BINKLEY, D., DAWN, D. L. et UEHLINGER, C. (2012). Vocabulary Normalization Improves IR-Based Concept Location. Proceedings of the 28th International Conference on Software Maintenance, vol. 41, pp. 588–591.

ANTONIOL, G., CANFORA, G., CASAZZA, G., LUCIA, A. D., et MERLO, E. (2002). Recovering Traceability Links Between Code and Documentation. IEEE Transactions on Software Engineering, vol. 28, pp. 970–983.

MALETIC, J. I. et COLLARD, M. L. (2009). Tql: A Query Language to Support Traceability. Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering. pp. 16–20

References

DE LUCIA, A., DI PENTA, M. et OLIVETO, R. (2010). Improving Source Code Lexicon via Traceability and Information Retrieval. IEEE Transactions on Software Engineering, vol. 37, pp. 205–226.

GUERROUJ, L., DI PENTA, M., GU´EH´ENEUC, Y.-G. et ANTONIOL, G. (2013b). An Experimental Investigation on the Effects of Context on Source Code Identifiers Splitting and Expansion. Empirical Software Engineering. Doi: 10.1016/S0164-1212(00)00029-7.

GUERROUJ, L., DI PENTA, M., ANTONIOL, G. et GU´EH´ENEUC, Y.-G. (2013a). TIDIER: An Identifier Splitting Approach using Speech Recognition Techniques. Journal of Software Evolution and Process, vol. 25, pp. 569–661.

DIT, B., GUERROUJ, L., POSHYVANYK, D. et ANTONIOL, G. (2011). Can Better Identifier Splitting Techniques Help Feature Location? Proceedings of the 19th International Conference on Program Comprehension. pp. 11–20.

GUERROUJ, L., GALINIER, P., GU´EH´ENEUC, Y.-G., ANTONIOL, G. et DI PENTA, M. (2012). TRIS: A Fast and Accurate Identifiers Splitting and Expansion Algorithm. Proceedings of the 19th Working Conference on Reverse Engineering. pp. 103–112.

MADANI, N., GUERROUJ, L., DI PENTA, M., GU´EH´ENEUC, Y.-G. et ANTONIOL, G. (2010). Recognizing Words from Source Code Identifiers using Speech Recognition Techniques. Proceedings of the 14th European Conference on Software Maintenance and Reengineering. pp. 68–77.

NEY, H. (1984). The Use of a One-stage Dynamic Programming Algorithm for Connected Word Recognition. IEEE Transactions on Acoustics Speech and Signal Processing, vol. 32, pp. 263–271.