CPC: Automatically Classifying and Propagating Natural Language Comments via Program Analysis

Juan Zhai, Xiangzhe Xu, Yu Shi, Guanhong Tao, Minxue Pan, Shiqing Ma,

Lei Xu, Weifeng Zhang, Lin Tan, Xiangyu Zhang

PURDUE UNIVERSITY

RUTGERS THE STATE UNIVERSITY OF NEW JERSEY

南京大學 NANJING UNIVERSITY

南京郵電大學 Nanjing University of Posts and Telecommunications

Motivation

Code comments provide abundant information which can be leveraged to help perform various software engineering tasks, such as bug detection, specification inference and code synthesis.

Developers are less motivated to write and update comments, making it infeasible and error-prone to leverage comments to facilitate software engineering tasks.

Provide Automation Support in Maintaining Comments

Information Propagation

- Program analysis techniques propagate information based on program semantics.

```
boolean y = true;
…
int x = y; //fault, detected by compiler for strong typed languages
```

  - If `x` = `y`, then `x` and `y` should have the same *data type*.

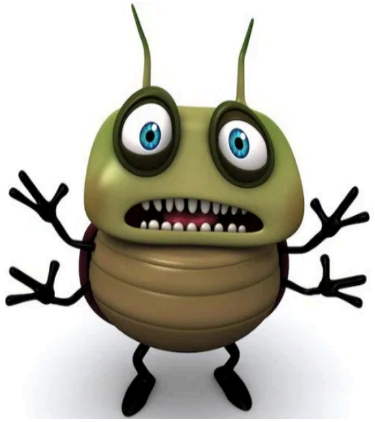
Can we propagate comments of `y` to `x`?

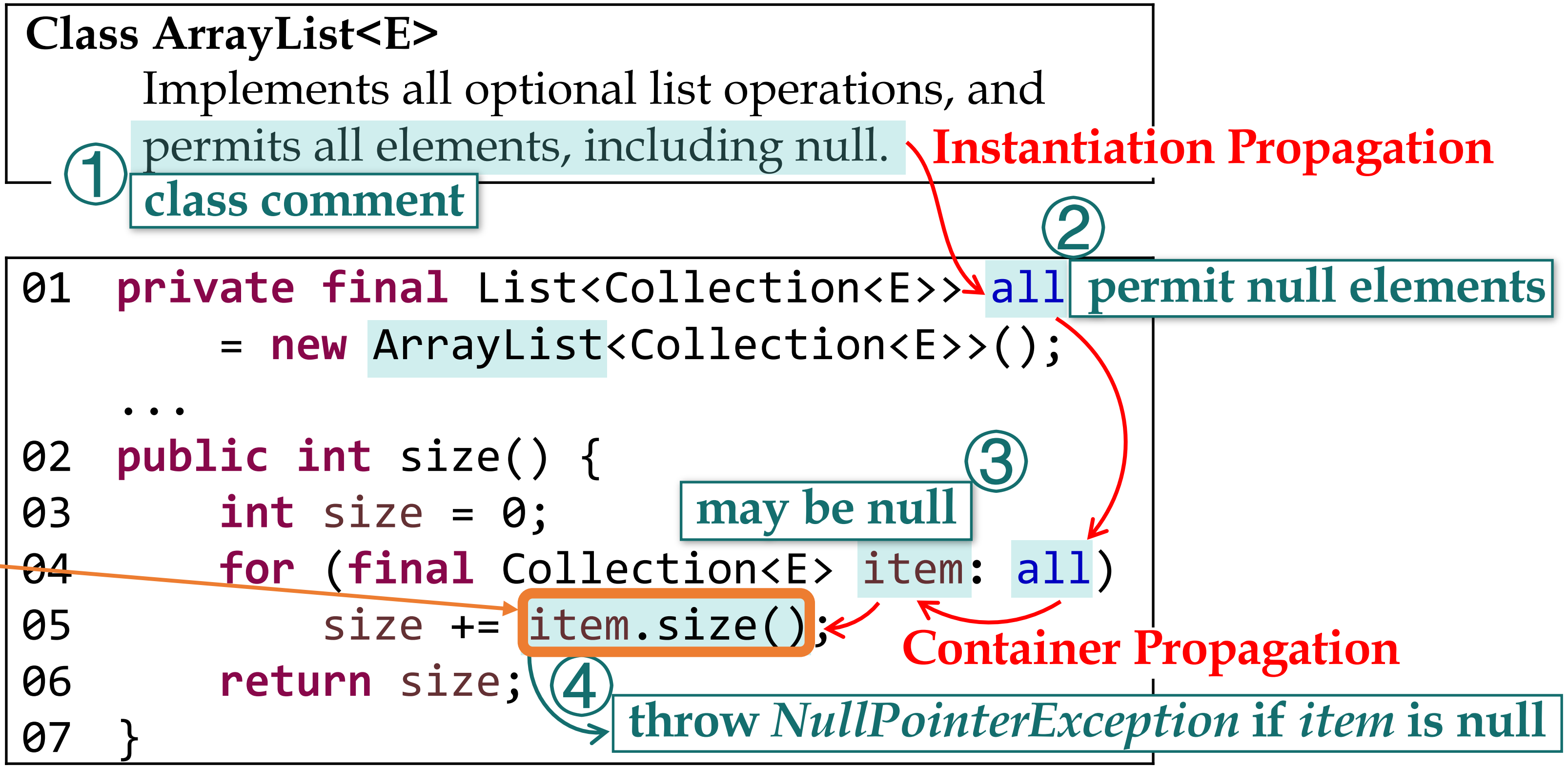
Idea

- Treat comments as first-class objects and leverage program analysis to derive, refine and propagate comments.
- Propagated comments provide additional semantic hints to enrich program analysis, like bug detection, especially for code without existing comments.

Code-comment Co-analysis

Code-comment Co-analysis Example





Class ArrayList<E>

Implements all optional list operations, and permits all elements, including null.

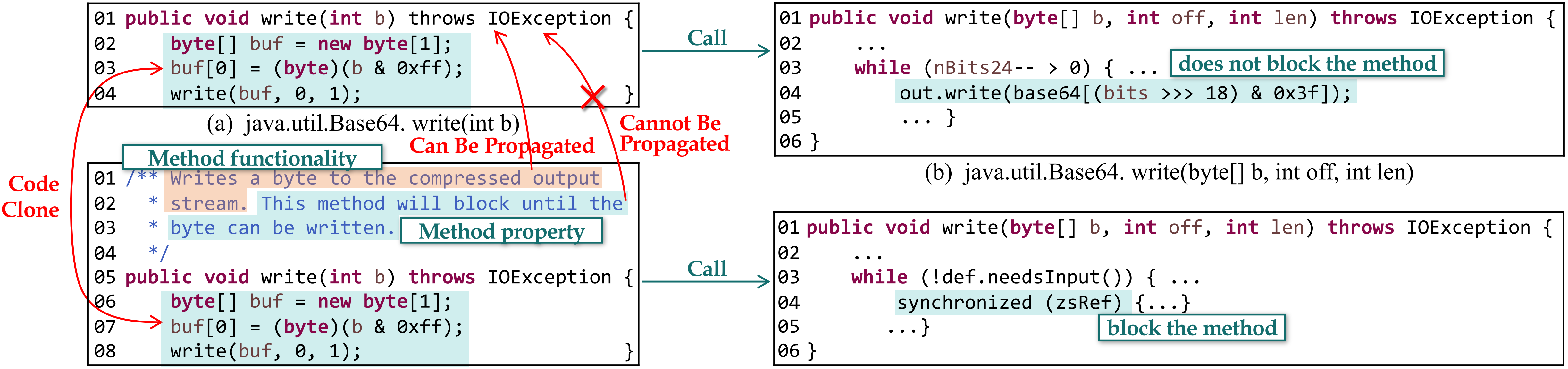
```
private final List<Collection<E>> all
        = new ArrayList<Collection<E>>();
    ...
public int size() {
    int size = 0;
    for (final Collection<E> item: all)
        size += item.size();
    return size;
}
```

① class comment
Instantiation Propagation
② permit null elements
③ may be null
Container Propagation
④ throw *NullPointerException* if *item* is null

Comment Classification Motivation

Q: Can we propagate arbitrary comments following a unified rule?

A: No! Software developers tend to comment on different content aspects of different code elements. Comments must be classified!



```
public void write(int b) throws IOException {
    byte[] buf = new byte[1];
    buf[0] = (byte)(b & 0xff);
    write(buf, 0, 1);                            }
```

(a) java.util.Base64. write(int b)

Can Be Propagated

Cannot Be Propagated

Call

```
public void write(byte[] b, int off, int len) throws IOException {
    ...
    while (nBits24-- > 0) { ...
        out.write(base64[(bits >>> 18) & 0x3f]);
        ... }
}
```

does not block the method

(b) java.util.Base64. write(byte[] b, int off, int len)

Code Clone

Method functionality

```
/** Writes a byte to the compressed output
  * stream. This method will block until the
  * byte can be written.
  */
public void write(int b) throws IOException {
    byte[] buf = new byte[1];
    buf[0] = (byte)(b & 0xff);
    write(buf, 0, 1);                            }
```

Method property

(c) java.util.zip.DeflataerOutputStream. write(int b)

Call

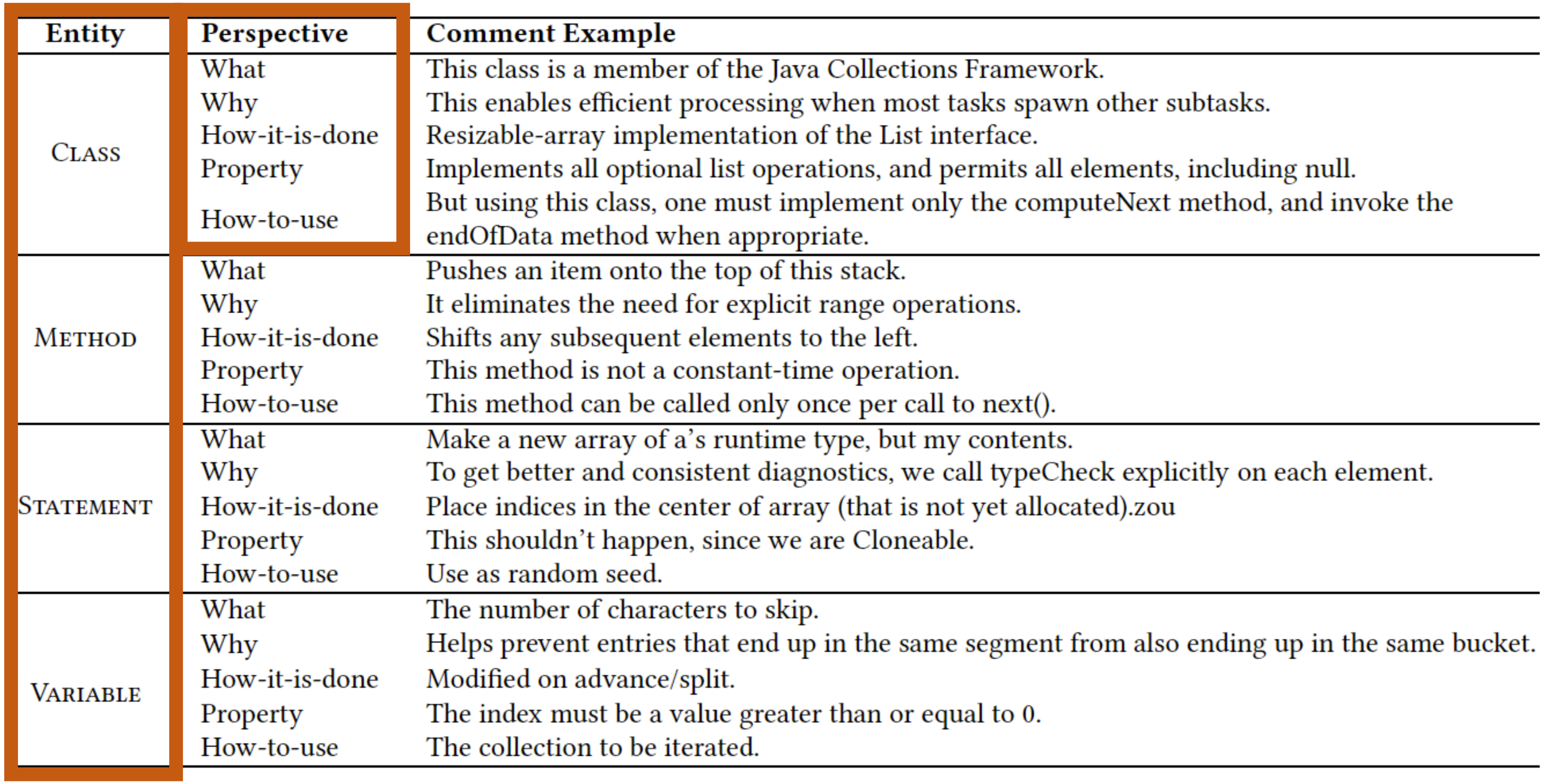
```
public void write(byte[] b, int off, int len) throws IOException {
    ...
    while (!def.needsInput()) { ...
        synchronized (zsRef) {...}
      ...}
}
```

block the method

(d) java.util.zip.DeflataerOutputStream.write(byte[] b, int off, int len)

The Taxonomy of Comments

- Different comments describe different code entities from different content perspectives



Entity	Perspective	Comment Example
Class	What	This class is a member of the Java Collections Framework.
	Why	This enables efficient processing when most tasks spawn other subtasks.
	How-it-is-done	Resizable-array implementation of the List interface.
	Property	Implements all optional list operations, and permits all elements, including null.
	How-to-use	But using this class, one must implement only the computeNext method, and invoke the endOfData method when appropriate.
Method	What	Pushes an item onto the top of this stack.
	Why	It eliminates the need for explicit range operations.
	How-it-is-done	Shifts any subsequent elements to the left.
	Property	This method is not a constant-time operation.
	How-to-use	This method can be called only once per call to next().
Statement	What	Make a new array of a's runtime type, but my contents.
	Why	To get better and consistent diagnostics, we call typeCheck explicitly on each element.
	How-it-is-done	Place indices in the center of array (that is not yet allocated).zou
	Property	This shouldn't happen, since we are Cloneable.
	How-to-use	Use as random seed.
Variable	What	The number of characters to skip.
	Why	Helps prevent entries that end up in the same segment from also ending up in the same bucket.
	How-it-is-done	Modified on advance/split.
	Property	The index must be a value greater than or equal to 0.
	How-to-use	The collection to be iterated.

Content Perspectives

- **What**: the functionality
  - *Pushes an item onto the top of this stack.*
- **How-it-is-done**: the implementation details
  - *Shits the element currently at that position and any subsequent elements to the right.*
- **Property**: properties of the subject like pre-conditions and post-conditions
  - *The index must be a value greater than or equal to 0.*
- **Why**: why the subject is provided or the design rationale of the subject
  - *Helps prevent entries that end up in the same segment from also ending up in the same bucket.*
- **How-to-use**: the expected set-up of using the subject like platforms
  - *But using this class, one must implement only the computeNext method.*

Taxonomy Construction

- Perform a large-scale study of comments to build the taxonomy
  - Comment Sampling
    - Randomly collected 5,000 comment units (sentences) from four frequently-used open-sourced libraries (i.e., JDK, Guava, Apache Commons Collections and Joda)
  - Coding Procedure
    - Four coders participated in manually categorizing comments
    - Each comment is assigned to two different coders to minimize subjectivity

Automatic Comment Classification

- Train classification models separately for perspectives and code entities

	Perspective			Code Entity		
	DT	RF	CNN	DT	RF	CNN
Precision	87.84%	87.78%	95.15%	97.39%	98.09%	89.33%
Recall	95.22%	91.39%	93.78%	99.27%	99.27%	75.28%

Three algorithms: Decision Tree (DT), Random Forest (RF), and Convolutional Neural Network (CNN)

The classifiers are ***effective*** in classifying comments.

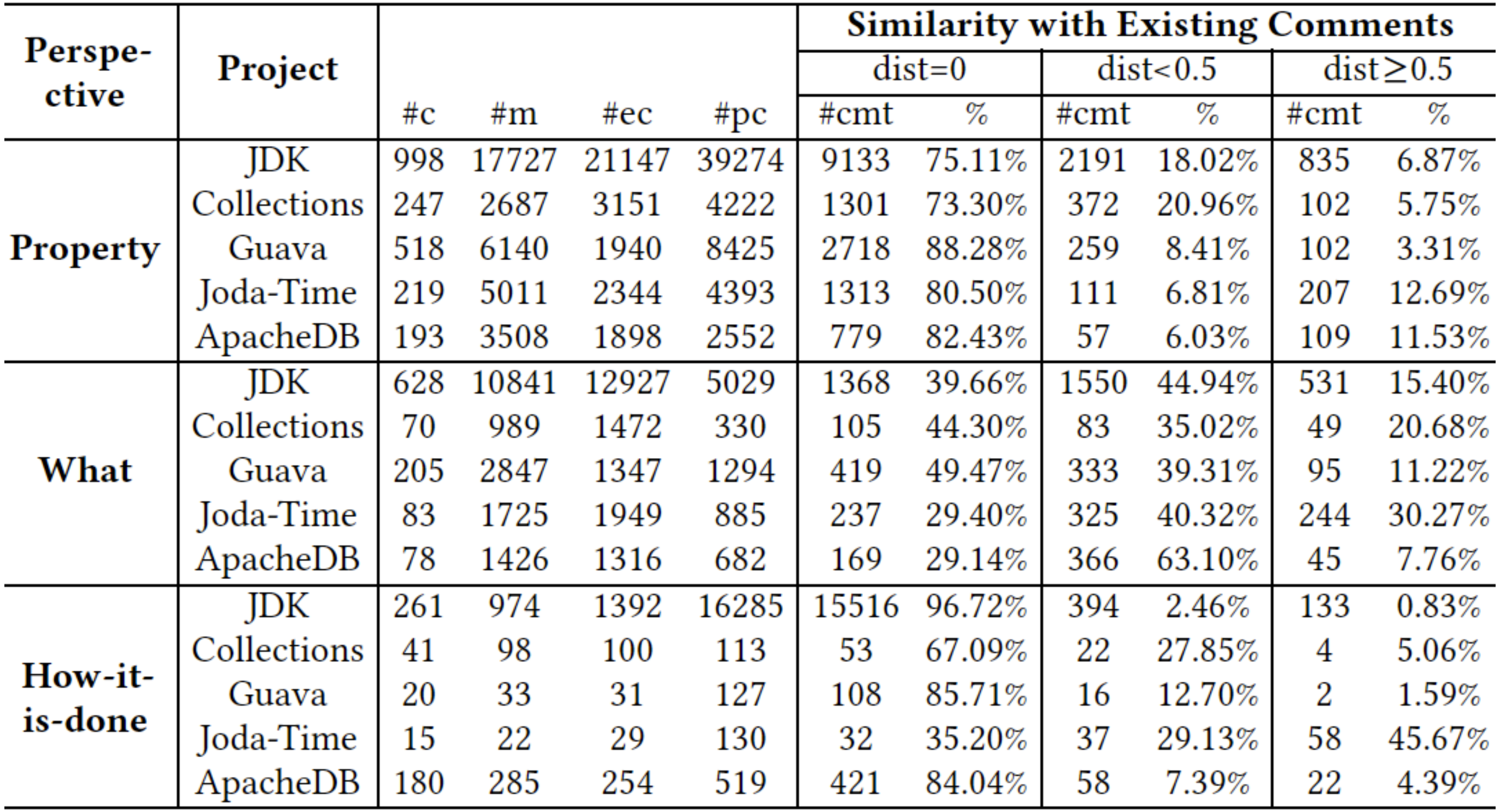
Comment Propagation via Program Analysis

- Comment propagation rules are derived from program semantics.
  - Certain and rigorous, generally applicable to all projects.
  - Different propagation rules for different code entities and different content perspectives.
  - Read our paper for details of rules.

Evaluation Setup

- Hardware
    - CPU: Intel® i7-8700K
    - RAM: 32GB
- Operating System
    - MacOS High Sierra 10.13.6
- JDK version: 8

Comment Propagation Summary



Perspective	Project					Similarity with Existing Comments					
						dist=0		dist<0.5		dist≥0.5	
		#c	#m	#ec	#pc	#cmt	%	#cmt	%	#cmt	%
Property	JDK	998	17727	21147	39274	9133	75.11%	2191	18.02%	835	6.87%
	Collections	247	2687	3151	4222	1301	73.30%	372	20.96%	102	5.75%
	Guava	518	6140	1940	8425	2718	88.28%	259	8.41%	102	3.31%
	Joda-Time	219	5011	2344	4393	1313	80.50%	111	6.81%	207	12.69%
	ApacheDB	193	3508	1898	2552	779	82.43%	57	6.03%	109	11.53%
What	JDK	628	10841	12927	5029	1368	39.66%	1550	44.94%	531	15.40%
	Collections	70	989	1472	330	105	44.30%	83	35.02%	49	20.68%
	Guava	205	2847	1347	1294	419	49.47%	333	39.31%	95	11.22%
	Joda-Time	83	1725	1949	885	237	29.40%	325	40.32%	244	30.27%
	ApacheDB	78	1426	1316	682	169	29.14%	366	63.10%	45	7.76%
How-it-is-done	JDK	261	974	1392	16285	15516	96.72%	394	2.46%	133	0.83%
	Collections	41	98	100	113	53	67.09%	22	27.85%	4	5.06%
	Guava	20	33	31	127	108	85.71%	16	12.70%	2	1.59%
	Joda-Time	15	22	29	130	32	35.20%	37	29.13%	58	45.67%
	ApacheDB	180	285	254	519	421	84.04%	58	7.39%	22	4.39%

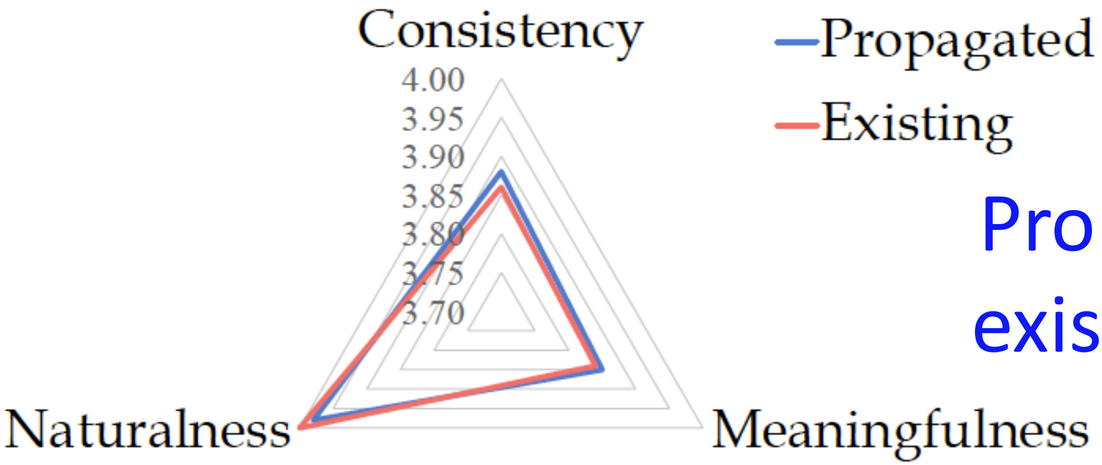
In the 5 projects, 41,573 new comments can be derived by propagation.

Comment Propagation Accuracy

- Randomly sampled 500 comments of each perspective and manually checked them
    - A propagated comment is accurate when it is consistent with source code.
    - CPC achieves 88% accuracy.

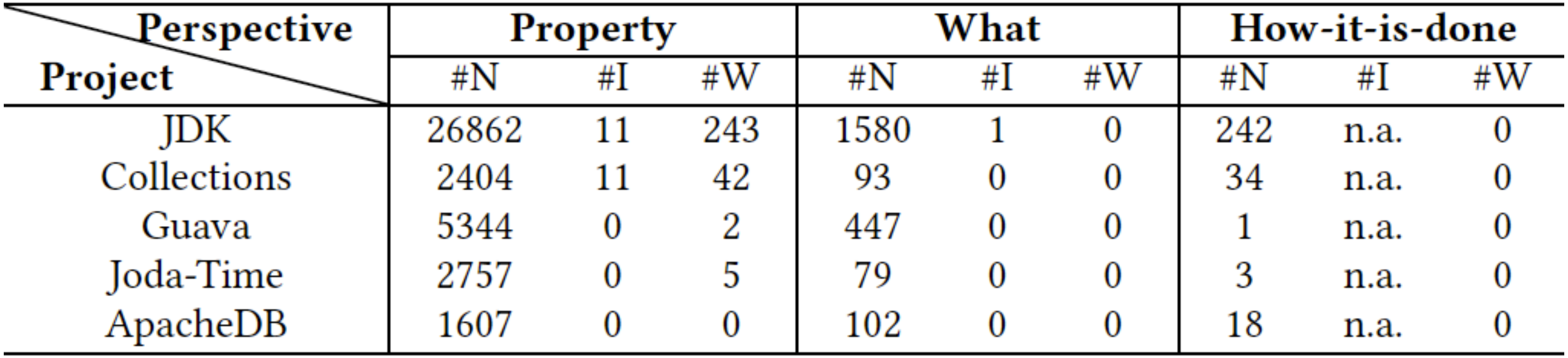
User Study: Usefulness in Helping Developers

- To avoid bias: propagated comments and existing comments are mixed
  - Meaningfulness: is a comment of high quality in helping developers understand code?
  - Consistency: is a comment consistent with code?
  - Naturalness: does a comment effectively convey information as a natural language sentence?



Propagated comments align well with existing comments in terms of quality.

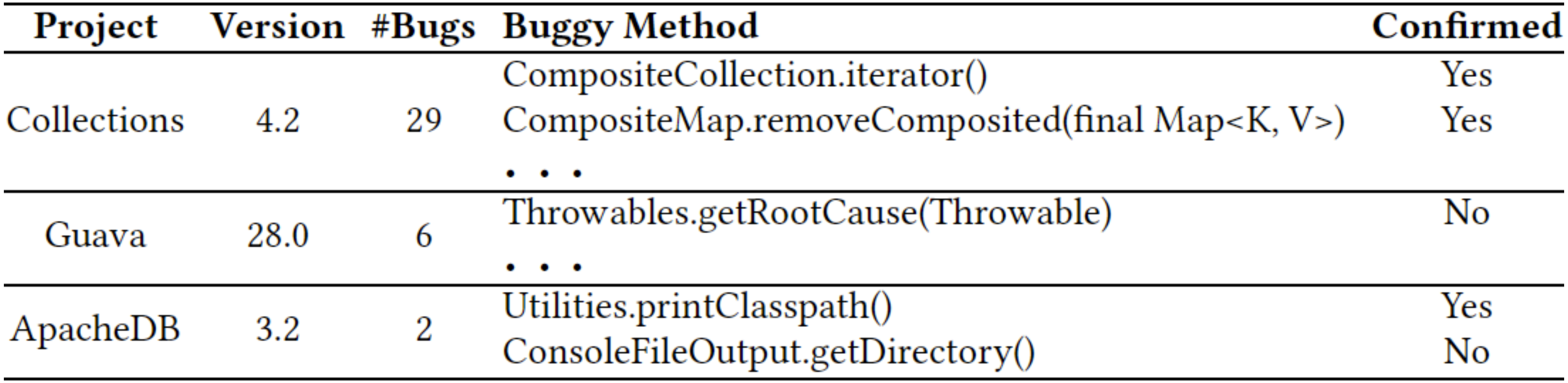
Effectiveness in Improving Comments



Perspective / Project	Property			What			How-it-is-done		
	#N	#I	#W	#N	#I	#W	#N	#I	#W
JDK	26862	11	243	1580	1	0	242	n.a.	0
Collections	2404	11	42	93	0	0	34	n.a.	0
Guava	5344	0	2	447	0	0	1	n.a.	0
Joda-Time	2757	0	5	79	0	0	3	n.a.	0
ApacheDB	1607	0	0	102	0	0	18	n.a.	0

- Precise functional comments are inferred for 87 native methods that have neither comments nor code.
- 12 incomplete comments and 292 wrong comments are identified.
- Developers confirmed and corrected some of the wrong existing comments.

Effectiveness in Bug Detection



Project	Version	#Bugs	Buggy Method	Confirmed
Collections	4.2	29	CompositeCollection.iterator()	Yes
			CompositeMap.removeComposited(final Map<K, V>)	Yes
			• • •	
Guava	28.0	6	Throwables.getRootCause(Throwable)	No
			• • •	
ApacheDB	3.2	2	Utilities.printClasspath()	Yes
			ConsoleFileOutput.getDirectory()	No

Detect and report 37 bugs, 30 of which have been confirmed and fixed by developers.

Related Work

- Comment Classification
  - Pascarella [MSR ’17] , Maalej [TSE ’13], Steidl [ICPC ’13], Monperrus [EMSE ’12], Haouari [ESEM ’11], Padioleau [ICSE ’09]
- Comment Generation
  - LeClair [ICML ’19], Hu [ICPC ’18], Iyer [ACL ’16], Jiang [ASE ’17], Allamanis [ICML ’16], McBurney [TSE ’16], Wong [ASE ’16, SANER ’15], Moreno [ICPC ’13], Rastkar [ICSM ’11], Sridhara [ICPC ’11, ASE ’10]
- Comment-Code Inconsistency Detection
  - Zhou [ICSE ’17], Tan [ICST ’12], Tan [ICSE ’11, SOSP ’07]

Conclusion

- We construct a comprehensive comment taxonomy from different perspectives with various granularity levels.
- We achieve a seamless synergy of comment analysis and program analysis:
  - Leverage program analysis to propagate comments.
  - Leverage comment analysis to facilitate program analysis.

https://setext.github.io/

Artifacts Available acm

Thank You