Identifier-Based Context-Dependent API Method Recommendation

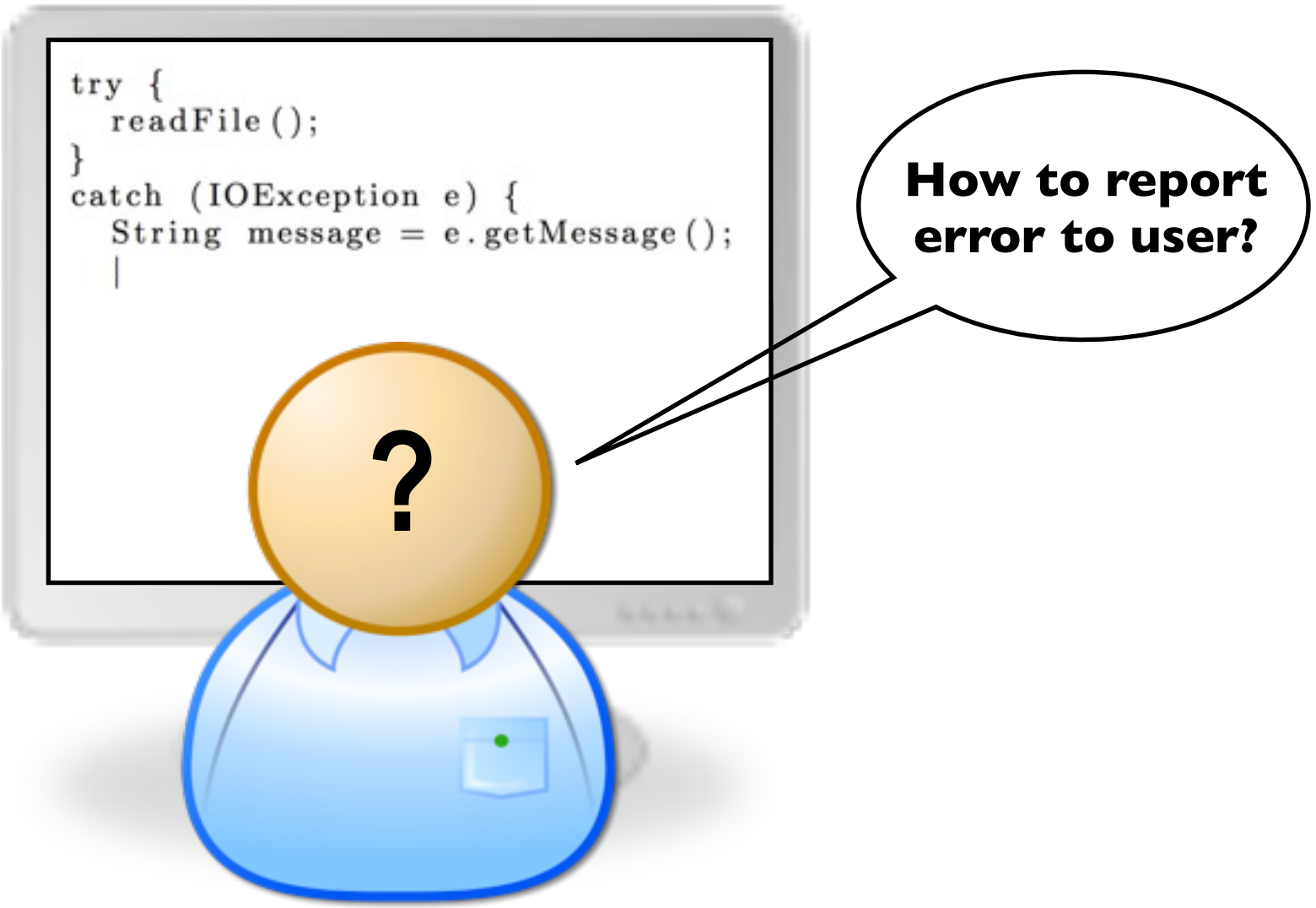
Lars Heinemann, Veronika Bauer,
Markus Herrmannsdörfer, Benjamin Hummel
Technische Universität München

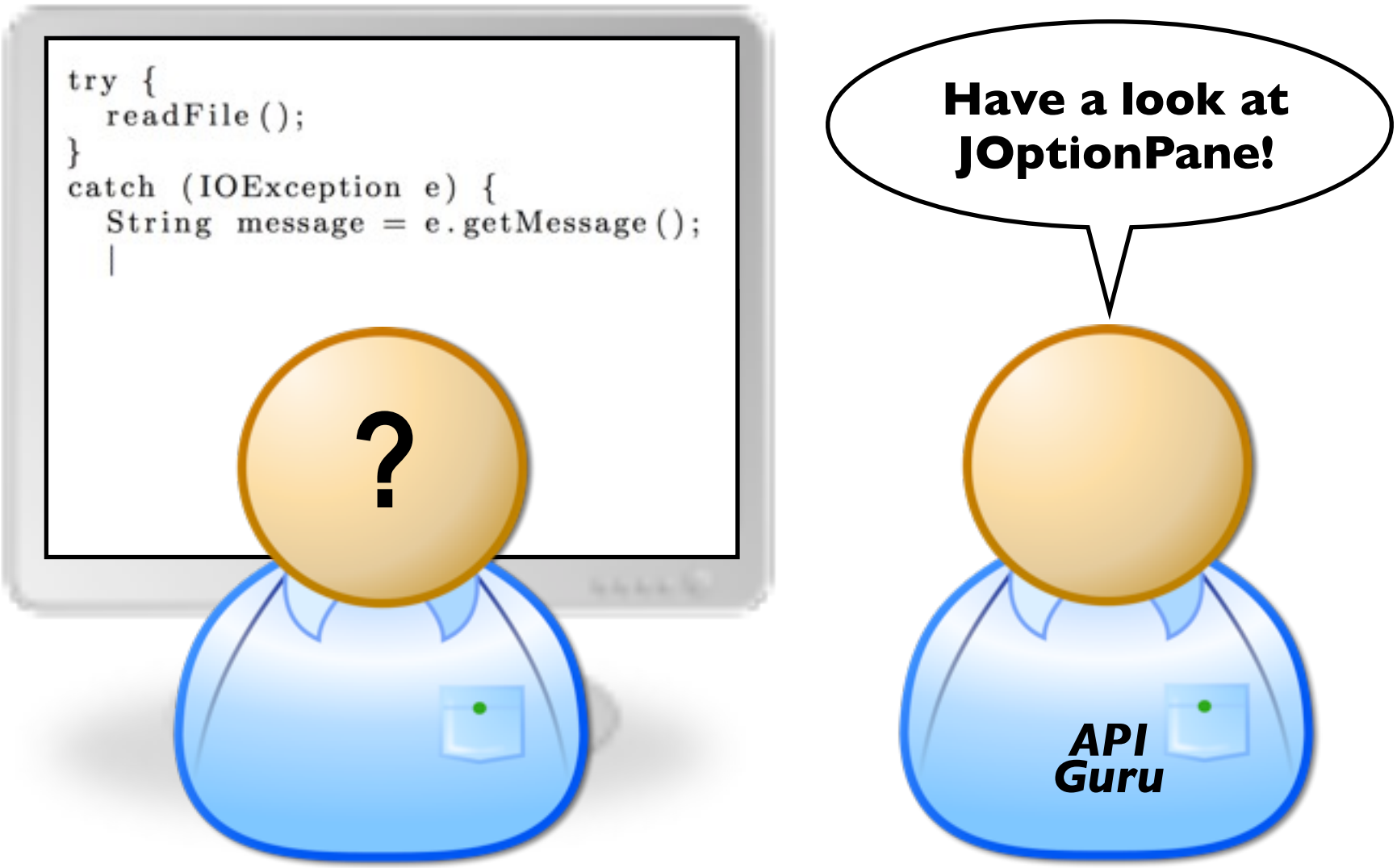
CSMR 2012, Szeged, Hungary

Software Libraries

- Popular way of reusing third-party software
- ***Challenge***: APIs often large and complex

Library	Version	API types
Apache Commons IO	2.0.1	98
Dom4j	2.0	159
JFreeChart	1.0.13	586
Spring Framework	2.0	1,680
Eclipse Platform	3.6	3,588
Java Standard API	1.7	4,024


try {
readFile();
}
catch (IOException e) {
String message = e.getMessage();
How to report error to user?
?


try {
readFile();
}
catch (IOException e) {
String message = e.getMessage();
?
Have a look at JOptionPane!
API Guru

API Recommendation Systems

- Approaches similar to shopping basket analysis

- Derive from context what may be useful API elements
- Existing approaches use structural context information

API Recommendation Systems (2)

- *Problem*: How to derive recommendations in case of
  - general purpose types
  - no preceding API usage in context

```
if (angle != getAngle()) {
  float angleDelta = angle - getAngle();
  super.setAngle(angle);
```

- Here: only general purpose types and simple getters/setters

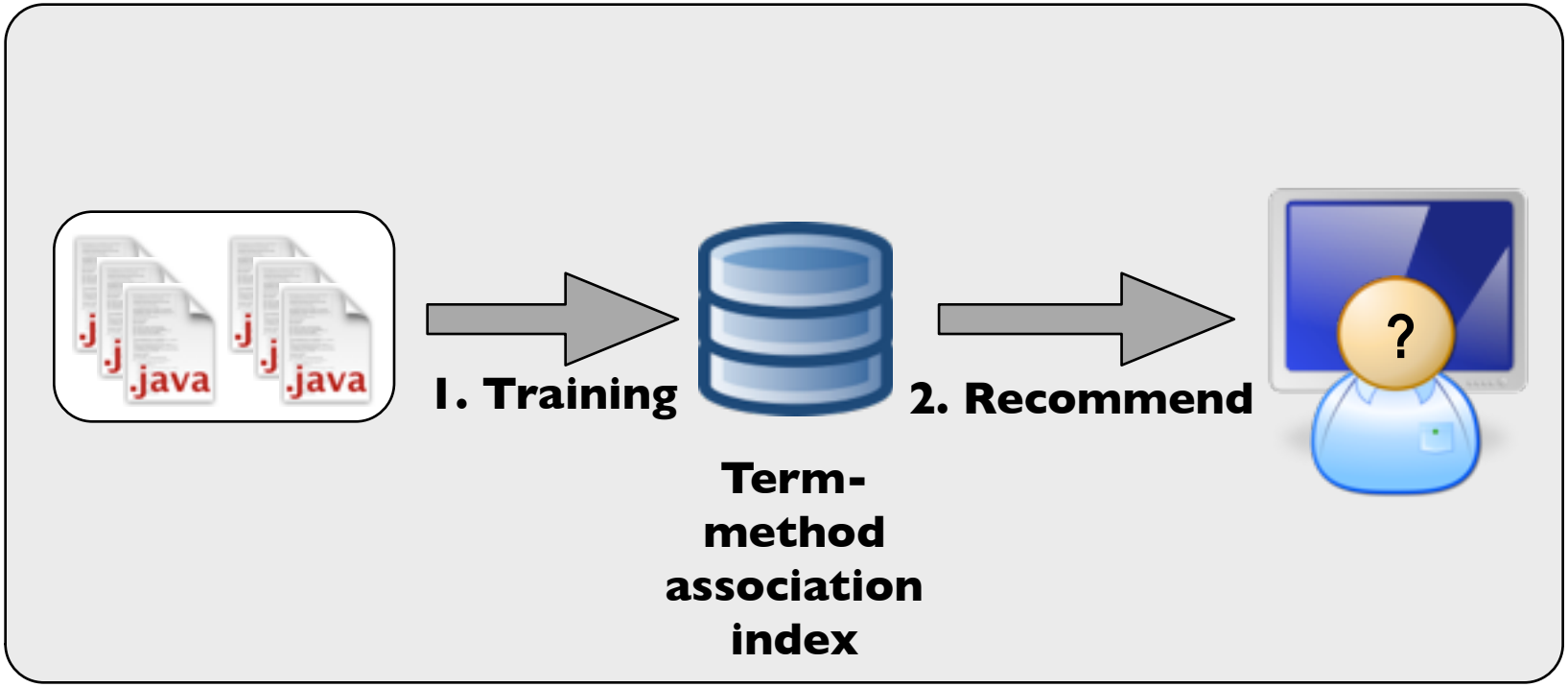
Proposed Approach - Basic idea

- Focus on identifiers
- Embody knowledge about developers‘ intent
- In the example: *trigonometry*

```
if (angle != getAngle()) {
   float angleDelta = angle - getAngle();
   super.setAngle(angle);
```

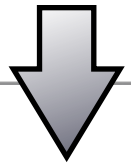
➪ Suggestion of trigonometric API functions useful

Proposed Approach - Two Phases



Training Phase

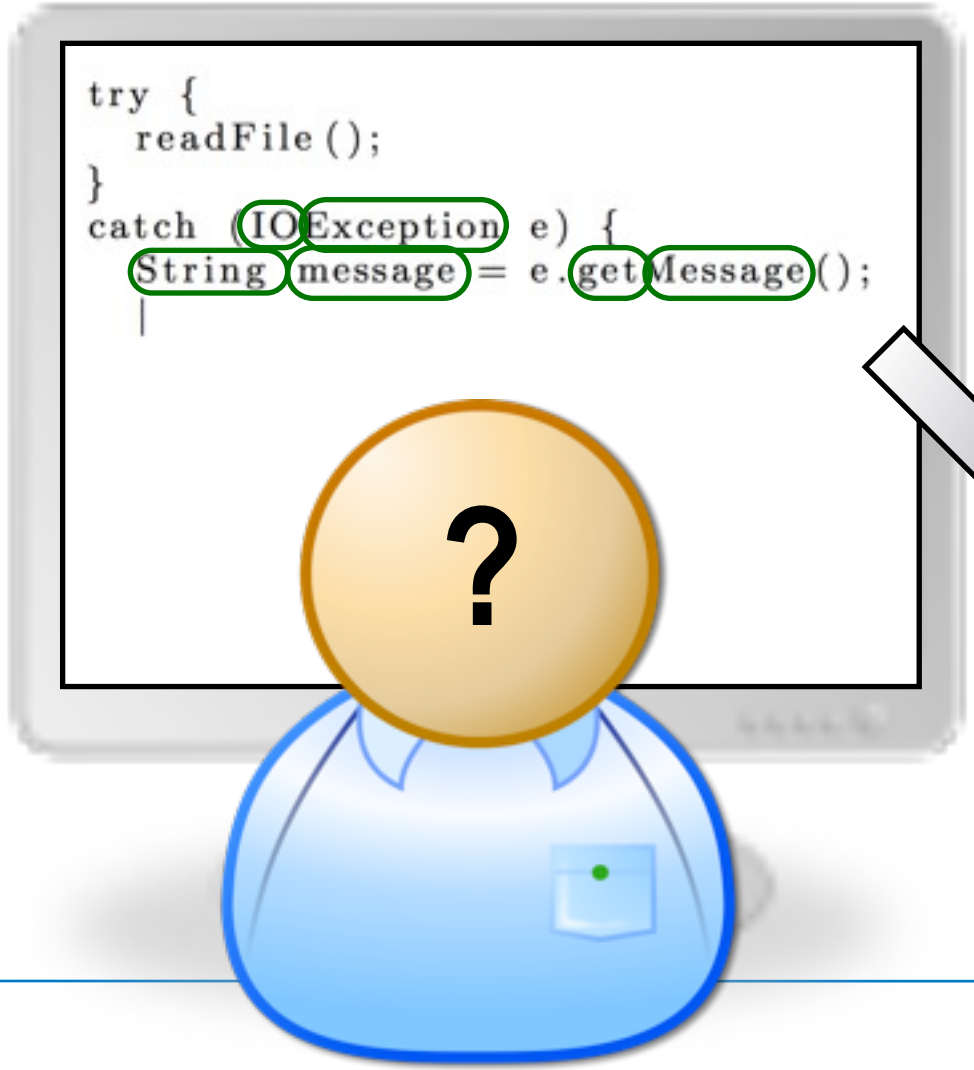
```
try {
   readFile();
}
catch (IOException e) {
   String message = e.getMessage();
   JOptionPane.showMessageDialog(null,
      message);
}
```



Identifier extraction, splitting, stemming

{io, except, string, messag, get}→
JOptionPane#showMessageDialog

Recommendation Phase



Identifier extraction, splitting, stemming

{io, except, string, messag, get}

Recommendation Phase (2)

- Find similar contexts in index
- Recommend associated methods
- Requires notion of similarity between sets of terms

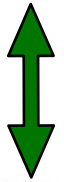
Jaccard Similarity

$$js(T_1, T_2) = \frac{|(T_1 \cap T_2)|}{|(T_1 \cup T_2)|}$$

$$js(\{file, input, read\}, \{file, write\}) = \frac{|\{file\}|}{|\{file, input, read, write\}|} = 0.25$$

Variation Points

```
try {
   readFile();
}
catch (IOException e) {
   String message = e.getMessage();
   JOptionPane.showMessageDialog(null,
      message);
}
```



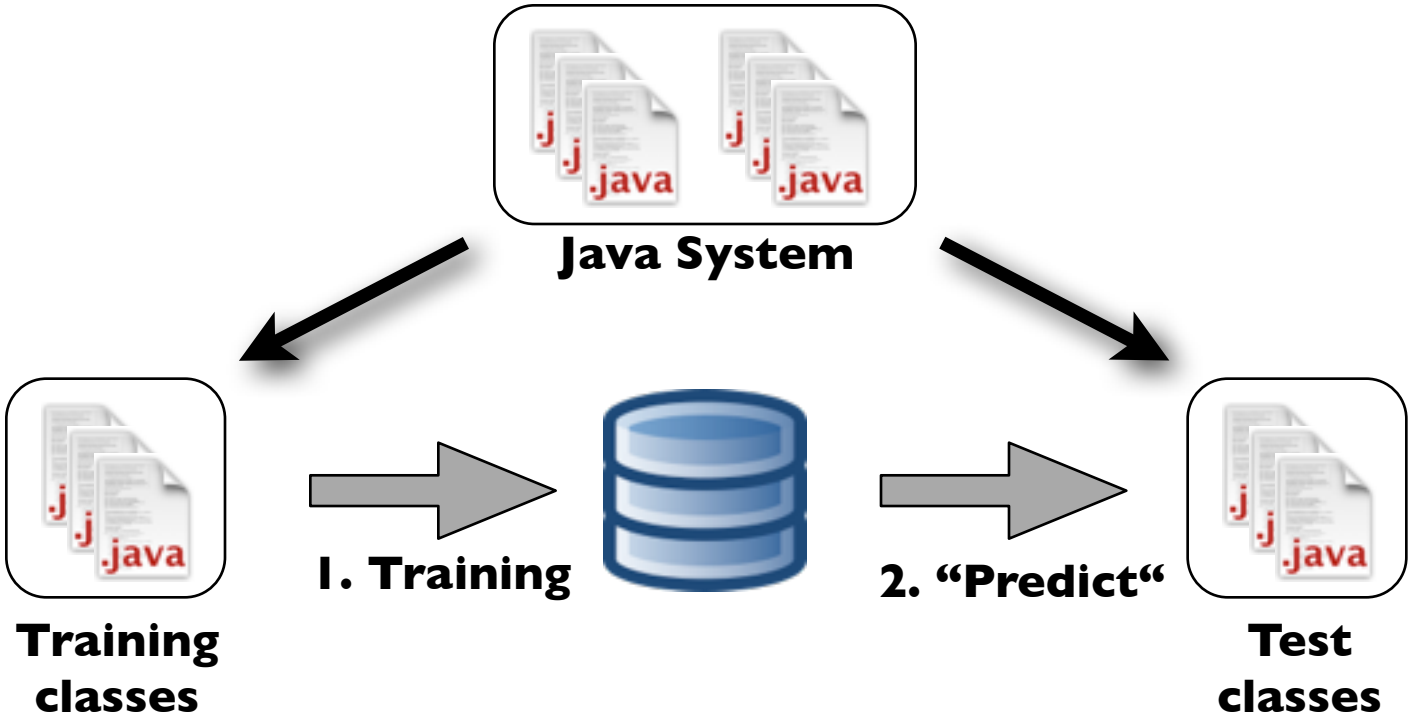
- *lookback*: How many identifiers are considered as context?
- *keywords*: Including keywords (e.g. `catch` indicates error handling)

Case Study: Research Questions

RQ 1: Influence of variation points

RQ 4: Comparison to method-usage based approach

Evaluation methodology



Evaluation methodology (2)

- "Remove" API method calls in test classes
- Use recommendation system to "guess" calls from context

```
try {
   readFile();
}
catch (IOException e) {
   String message = e.getMessage();


}
```

Case Study: Design and Procedure

- Recommend 5 API methods for each query
- *Recommendation rate*: How often is actual method among recommendations?
- *Baseline rate*: Always recommend 5 top-used methods

Study Objects

API	System	API Calls	API Methods	Baseline
Java	DrJava	21,090	2,026	11.8%
	FreeMind	8,725	1,439	12.9%
	HSQLDB	9,735	1,100	24.0%
	JabRef	21,350	1,691	18.1%
	JEdit	17,341	1,934	10.5%
	SoapUI	24,659	2,500	11.3%

Results: Influence of parameters

- Average recommendation rate (RR) for different ***lookbacks***

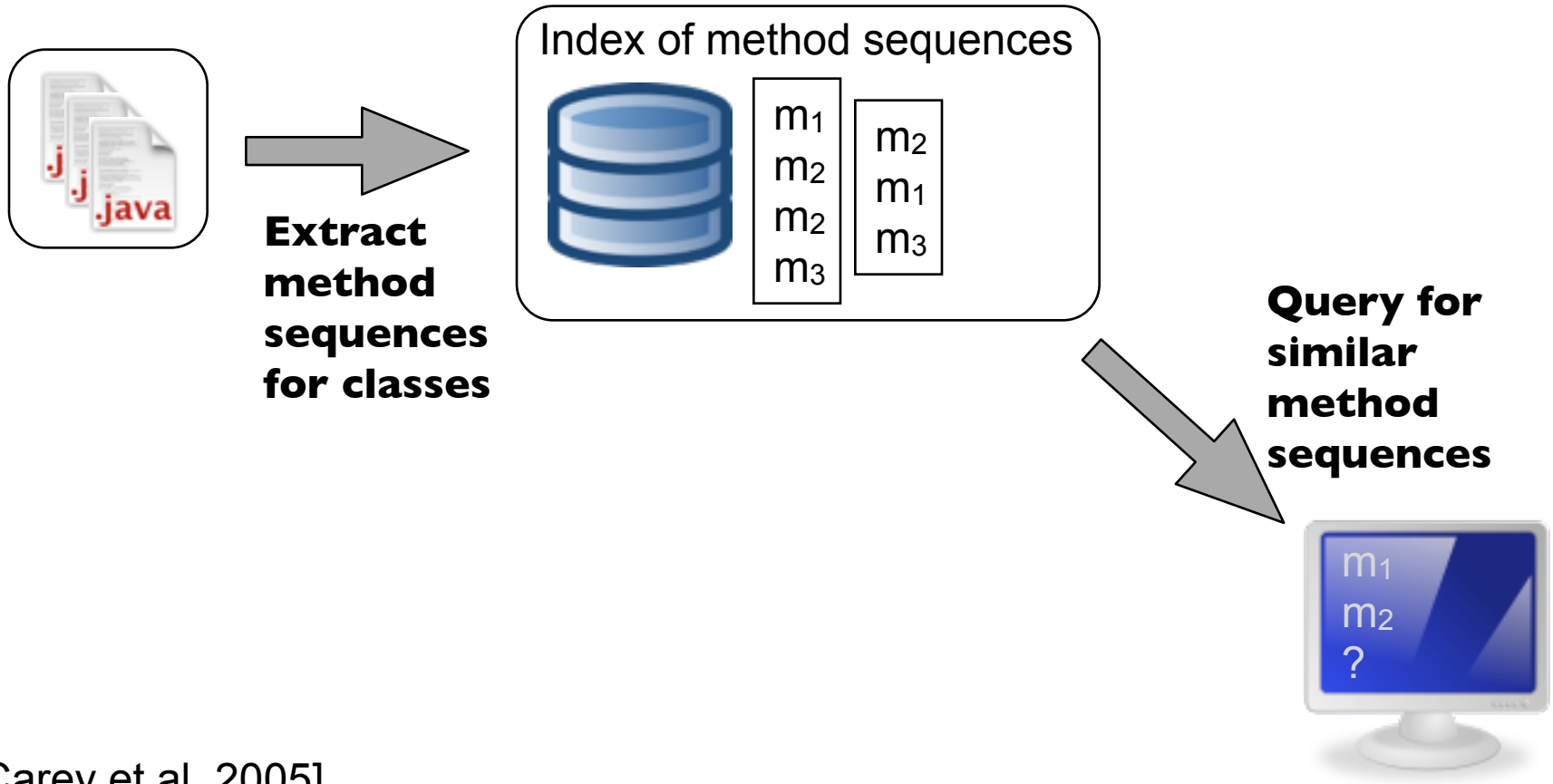
Lookback	Avg. RR
1	33.4%
2	40.5%
3	40.8%
4	**41.3%**
5	40.7%
6	40.5%

Results: Influence of parameters

- Recommendation rate for ***keywords*** excluded vs. included

System	Keywords	
	excluded	*included*
DrJava	40.5%	44.3%
FreeMind	39.0%	43.1%
HSQLDB	35.9%	41.6%
JabRef	47.0%	52.1%
JEdit	39.6%	45.1%
SoapUI	45.6%	51.2%
Average	**41.3%**	**46.2%**

Interlude: Rascal in a Nutshell



[McCarey et al. 2005]

“Applicability“ of Approaches

- Approaches need context to be able to recommend
  - Rascal: at least one preceding method call
  - Our approach: identifiers according to lookback, within same method body

Results: Comparison to Rascal

- RR_{appl} : Recommendation rate regarding *applicable* cases
- RR_{glob} : Recommendation rate regarding all cases (*global*)

Approach	Appl.	RR_{appl}	RR_{glob}
Ours	78.5%	46.2%	**36.1%**
Rascal	97.5%	27.9%	**27.2%**

Discussion

- "Correct" method recommended in almost half of the cases
- 5.5% - 17.0% more of the cases compared to Rascal
- Performance: Index creation: 30s-231s, average query: 5ms-26ms
- We also have an Eclipse integration:

Conclusion & Future Work

- Identifiers good alternative to structure-based approaches
- Hybrid approaches may achieve even better results
- Next step: User evaluation with developers

Thank you.
Questions?