

# MAPO: Mining and Recommending API Usage Patterns

Authors: Hao Zhong, Tao Xie, Lu Zhang, Jian Pei and Hong Mei  
Presented by: Wentao Fan

# Introduction

- Most API libraries are difficult to use
- Programmers struggle with choosing and organizing proper API methods
- Programmers may need to browse a large number of code snippets(to locate snippets with relevant usage) with given API methods(e.g. Google, Strathcona).



# Problem Statement

- Existing API methods are often complex and not well documented
- Returned code snippets are often large in number, and make it hard for programmers to locate useful ones.



# Proposed Tool

- **MAPO** (Mining API usage Pattern from Open source repositories) has been developed for mining API usage patterns automatically.
- **Pattern mining**: to cluster code snippets exhibiting different usages into different clusters
- **Pattern recommendations**: uses mined patterns as an index for their associated code snippets



# Example

- Find one method *appendToGroup*
- Use “appendToGroup lang:java” to query Google code search and it returns 151 code snippets
- Both snippets are put near the bottom of the returned list
- the first snippet is put as the 84th of the snippet list, and the second one is put as the 104th of the snippet list

```
public class DEditorActionContributor ... {  
    public void contributeToMenu(IMenuManager menu) {  
        super.contributeToMenu(menu);  
        IMenuManager editMenu = menu.findMenuUsingPath(IWorkbenchActionConstants.M_EDIT);  
        if(editMenu != null ){  
            editMenu.add(new Separator());  
            editMenu.appendToGroup("additions", fToggleInsertModeAction);  
        }  
        ...  
    }  
}
```

```
public class RubyEditorActionContributor ... {  
    public void contributeToMenu(IMenuManager menuManager) {  
        ...  
        IMenuManager gotoMenu = menu.findMenuUsingPath("navigate/goTo");  
        if(gotoMenu != null ){  
            gotoMenu.add(new Separator("additions2"));  
            gotoMenu.appendToGroup("additions2", fGotoMatchingBracket);  
        }  
        ...  
    }  
}
```

**Fig. 1.** Code snippets of “appendToGroup” returned by Google code search

# Example

- MAPO adopts a frequent subsequence miner to mine usage patterns from the code snippets in the cluster
- From each of the two clusters, MAPO acquires one usage pattern
- Mined patterns are usually much fewer than code snippets, so that locating can be more effective
- MAPO associates “Pattern1” in Figure 3 to the code snippets in Figure 1

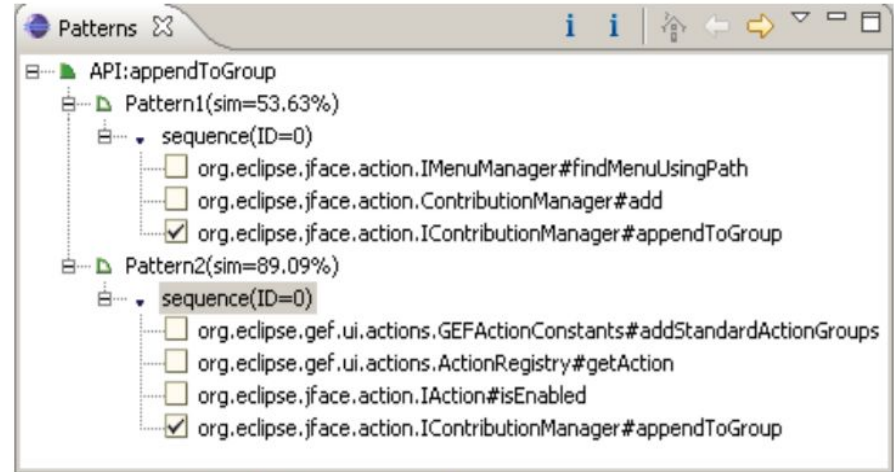


Fig. 3. Pattern index of “appendToGroup”

# Related Works

- **Recommending code snippets.** Evaluation (Section Experimental Study) shows that the mined patterns help programmers locate useful code snippets more effectively than approaches that recommend raw code snippets
- **Mining API properties.**
  - Category 1: mine association rules among software artifacts
  - Category 2: mine frequent call sequences from API client code or traces
  - Category 3: mine automata from API client code or traces



# Approach

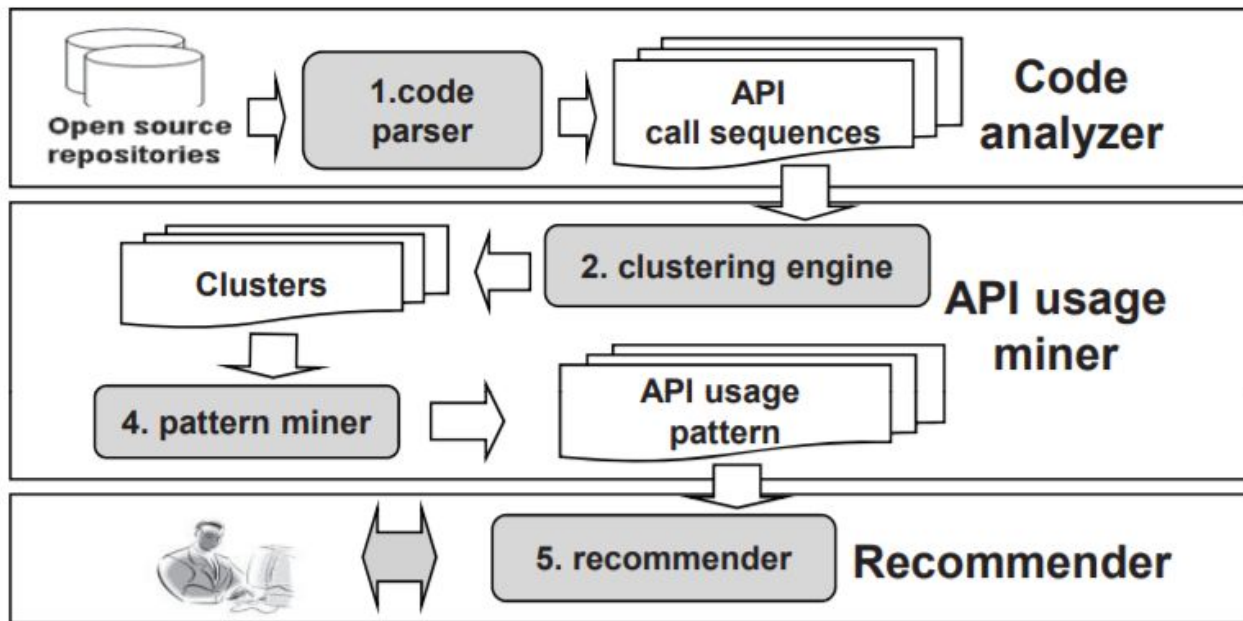


Fig.4. Overview of MAPO



# Approach: Source Code Analyzer

- **Collecting third-party API method calls**

- the corresponding call sequence of statement

*getGraphicalViewer().setRootEditPart( new ScalableRootEditPart())* is as follows:

@new org.eclipse.gef.editparts.ScalableRootEditPart

@org.eclipse.gef.ui.parts.GraphicalEditor#getGraphicalViewer

@org.eclipse.gef.EditPartViewer#setRootEditPart

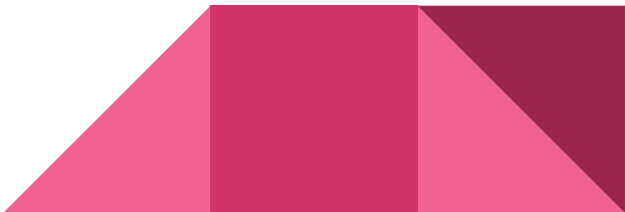


# Approach: Source Code Analyzer


- **Dealing with conditional statements**

- 6 possible API method call sequences: {i1, i2, i5}, {i1, i2, i6}, {i1, i2, i3, i5}, {i1, i2, i3, i6}, {i1, i2, i4, i6}, and {i1, i2, i4, i5}.

```
public void fun(boolean cond1, boolean cond2, boolean cond3){
    i1;
    i2;
    if(cond1)
        if(cond2) i3
    else i4;
    if(cond3) i5;
        else i6;
}
```



# Approach: Source Code Analyzer

- **Selecting a subset of sequences**
    - Method overweight: Different methods may contain different numbers of conditional statements.
    - Common-path overweight: common path objects may be called repeatedly.
  - **Inlining non-third-party methods**
    - Issue: A single method may not contain all the involved third-party API methods of an API usage scenario
    - When constructing the API method call sequences of  $m$ , inline the API method call sequences of each non-third-party method  $m'$  called by  $m$ .
    - When  $m$  and  $m'$  are in the same class, MAPO traverses the parser tree of the class for  $m$ 's API method call sequences
    - When not in the same class, MAPO resolves the declaring class of  $m$  and then finds the declaring class's source file for  $m$ 's method body
    - Iteration
- 

# Approach: API Usage Miner

- **Clustering API method call sequences**
  - In both code snippets in Figure 1, *appendToGroup* is used with *findMenuUsingPath* and *add*, and the API method call order is *findMenuUsingPath*→ *add*→ *appendToGroup*.
  - Observation: Among snippets, similar method/class names tend to exhibit the same usage
  - When calculating the similarity between names, split them into words

```
public class DEditorActionContributor ... {  
    public void contributeToMenu(IMenuManager menu) {  
        super.contributeToMenu(menu);  
        IMenuManager editMenu = menu.findMenuUsingPath(IWorkbenchActionConstants.M_EDIT);  
        if(editMenu != null ){  
            editMenu.add(new Separator());  
            editMenu.appendToGroup("additions", fToggleInsertModeAction);  
        }  
    }  
    ...  
}
```

```
public class RubyEditorActionContributor ... {  
    public void contributeToMenu(IMenuManager menuManager) {  
        ...  
        IMenuManager gotoMenu = menu.findMenuUsingPath("navigate/goTo");  
        if(gotoMenu != null ){  
            gotoMenu.add(new Separator("additions2"));  
            gotoMenu.appendToGroup("additions2", fGotoMatchingBracket);  
        }  
    }  
    ...  
}
```

**Fig. 1.** Code snippets of “appendToGroup” returned by Google code search

# Approach: API Usage Miner

- **Called API methods**

- Definition of the similarities between API method call sequences
- $I_1$  and  $I_2$  are the corresponding sets of API methods appearing in the two sequences.
- Numerator: The number of calls appearing in both sets
- Denominator: The number of calls appearing in either sets

$$\text{sim}(s_1, s_2) = \frac{\# \text{ of API calls in } I_1 \cap I_2}{\# \text{ of API calls in } I_1 \cup I_2}$$



# Approach: API Usage Miner

- **Mining API patterns**

- Minimum user-defined support: the number of API method call sequences that contain the patterns
- In each cluster (C), the support of an API method call sequence (s) is defined as follow:

$$\text{support}(s) = \frac{\# \text{ of API call sequences with } s}{\# \text{ of API call sequences in } C}$$



# Approach: API Usage Recommender

- Can use the mined patterns as an index to locate snippets.
- The rank of a pattern is the average similarity of the supporting snippets to the current programming task.
- Use the method names and the class names to calculate the similarity
- Use the returned patterns as an index to locate snippets

The screenshot displays the MAPO recommender interface within an IDE. It features several key components and annotations:

- API of interest:** Points to the `appendToGroup` method in the source code.
- Clicked sequence:** Points to the `appendToGroup` method in the source code.
- Contexts of methods:** Points to the `appendToGroup` method in the source code.
- Pattern:** Points to the `Pattern` class in the `Patterns` list.
- Pattern rank:** Points to the `Pattern rank` column in the `Patterns` list.
- Whether the API is invoked in this sample:** Points to the `Invoked` column in the `Support Samples` table.
- Similarity to the current programming context:** Points to the `Similarity` column in the `Support Samples` table.
- Methods with the clicked sequence Highlighted with background color:** Points to the `appendToGroup` method in the `Support Samples` table.
- Methods without the clicked sequence:** Points to the `buildContextMenu` method in the `Support Samples` table.
- Source code of selected method:** Points to the `buildContextMenu` method in the `Source Code` window.

className	methodName	Signature	Similarity	Invoked
ZenKitContextMenuProvider	buildContextMenu	void-IMenuManager	93.61%	Invoked
PatternContextMenuProvider	buildContextMenu	void-IMenuManager	86.66%	Invoked
FlowEditorContextMenuProvider	buildContextMenu	void-IMenuManager	81.11%	Invoked
EntityRelationContextMenuProvider	buildContextMenu	void-IMenuManager	92.22%	Invoked
SearchSheetContextMenuProvider	buildContextMenu	void-IMenuManager	92.22%	Invoked

```
public void buildContextMenu ( IMenuManager manager ) {
    GEFActionConstants.addAction( manager );
    IAction action;
    action = getActionRegistry().getAction( ZenModelerConstants.ZEN_KIT_ZOOM_IN );
    if ( action.isEnabled() )
        manager.appendToGroup( GEFActionConstants.GROUP_EDIT, action );
}
```

Fig. 5. MAPO recommender with annotations

# Evaluation: Experimental Study

- **Setup**

- Apply 20 open source projects which use **GEF**(Graphical Editing Framework) to develop graphical editors as a code repository.
- **MAPO** extracted API method call sequences and built clusters of these sequences using the technique presented in API usage miner.
- **Strathcona** is able to locate a set of relevant code snippets from a code repository.
- Restrict **Google search** scope to the same projects as MAPO.

**Table 1.** Projects used to mine patterns

Project	Project source	LOC	#classes	#methods
Work flow	TU Berlin	10125	101	1017
Net Editor	TU Berlin	2867	35	359
Sequence Editor	TU Berlin	3921	46	486
Visual OCL	TU Berlin	11967	134	1077
PetriEditor	TU Berlin	3248	44	375
jLibrary (Client)	SourceForge	46213	503	3455
Green UML	SourceForge	10652	146	1151
Quantum	SourceForge	2380	33	225
GanttRCP	SourceForge	3760	72	510
OpenWFE (IDE)	SourceForge	9952	178	954
Jupe	SourceForge	8100	109	665
Schema Viewer	SourceForge	3358	48	338
Janus	SourceForge	1952	19	132
ZEN-kit	University of California	3991	151	314
SimpleGEF	Bonevich	851	20	120
cvsgrapher	Bonevich	1706	29	179
GEF tutorial	EclipseTeam	837	19	122
GEF example	EclipseTeam	1299	22	155
Hello GEF	EclipseTeam	1042	18	144
OAW sample	Eclipse GMT	12777	203	1196
Total		140998	1930	12974



# Evaluation: Experimental Study

- **Quantitative Comparison**

- Prepared 13 programming tasks. In each task, use the first API method call and the programming context in the example to query the three tools
- Column “Total num. of items” lists the returned items from each query
- Sub-columns “Strat.” and “Google” list the number of snippets returned by Strathcona and Google code search respectively

**Table 2.** Comparison of Strathcona, Google code search, and MAPO

Example	First matched snippet			Second matched snippet			Total num. of items		
	Strat.	Google	MAPO	Strat.	Google	MAPO	Strat.	Google	MAPO
example 1	<b>5</b>	1	1	n/a	2	2	10	8	(2)
example 2	1	1	1	2	<b>n/a</b>	2	10	7	(1)
example 3	1	<b>3</b>	1	<b>5</b>	<b>4</b>	2	10	12	(4)(2)
example 4	n/a	4	n/a	n/a	10	n/a	10	11	n/a
example 5	1	<b>7</b>	2	3	<b>13</b>	3	10	33	(2)
example 6	n/a	9	n/a	n/a	11	n/a	10	33	n/a
example 7	2	<b>4</b>	2	<b>n/a</b>	<b>10</b>	3	10	39	(2)
example 8	n/a	n/a	n/a	n/a	n/a	n/a	10	18	(1)
example 9	<b>n/a</b>	<b>3</b>	1	<b>n/a</b>	<b>4</b>	2	10	28	(2)
example 10	1	1	1	2	2	2	10	16	(1)
example 11	<b>2</b>	<b>10</b>	1	<b>n/a</b>	<b>15</b>	2	10	39	(2)
example 12	<b>n/a</b>	1	1	<b>n/a</b>	2	2	10	27	(1)
example 13	2	2	2	3	<b>5</b>	3	10	70	(2)(1)

# Evaluation: Experimental Study

- **Quantitative Comparison(1st matched)**
  - **Strathcona** always returns 10 snippets
  - **Google code search** returns much fewer snippets than expected
    - Restrict search scope for fair
    - Can filter out snippets that match given keywords
  - **MAPO** relies on mined patterns to achieve a similar goal

**Table 2.** Comparison of Strathcona, Google code search, and MAPO

Example	First matched snippet			Second matched snippet			Total num. of items		
	Strat.	Google	MAPO	Strat.	Google	MAPO	Strat.	Google	MAPO
example 1	<b>5</b>	1	1	n/a	2	2	10	8	(2)
example 2	1	1	1	2	<b>n/a</b>	2	10	7	(1)
example 3	1	<b>3</b>	1	<b>5</b>	<b>4</b>	2	10	12	(4)(2)
example 4	n/a	4	n/a	n/a	10	n/a	10	11	n/a
example 5	1	<b>7</b>	2	3	<b>13</b>	3	10	33	(2)
example 6	n/a	9	n/a	n/a	11	n/a	10	33	n/a
example 7	2	<b>4</b>	2	<b>n/a</b>	<b>10</b>	3	10	39	(2)
example 8	n/a	n/a	n/a	n/a	n/a	n/a	10	18	(1)
example 9	<b>n/a</b>	<b>3</b>	1	<b>n/a</b>	<b>4</b>	2	10	28	(2)
example 10	1	1	1	2	2	2	10	16	(1)
example 11	<b>2</b>	<b>10</b>	1	<b>n/a</b>	<b>15</b>	2	10	39	(2)
example 12	<b>n/a</b>	1	1	<b>n/a</b>	2	2	10	27	(1)
example 13	2	2	2	3	<b>5</b>	3	10	70	(2)(1)

# Evaluation: Experimental Study

- **Quantitative Comparison(1st matched)**
  - **MAPO** often requires programmers to check fewer snippets for the first match than **Strathcona**
  - **MAPO** requires programmers to check fewer snippets for the first match than **Google code search**
  - **MAPO** uses patterns as an index for snippets, and it requires less effort to locate the 1st match than the other two tools

**Table 2.** Comparison of Strathcona, Google code search, and MAPO

Example	First matched snippet			Second matched snippet			Total num. of items		
	Strat.	Google	MAPO	Strat.	Google	MAPO	Strat.	Google	MAPO
example 1	<b>5</b>	1	1	n/a	2	2	10	8	(2)
example 2	1	1	1	2	<b>n/a</b>	2	10	7	(1)
example 3	1	<b>3</b>	1	<b>5</b>	<b>4</b>	2	10	12	(4)(2)
example 4	n/a	4	n/a	n/a	10	n/a	10	11	n/a
example 5	1	<b>7</b>	2	3	<b>13</b>	3	10	33	(2)
example 6	n/a	9	n/a	n/a	11	n/a	10	33	n/a
example 7	2	<b>4</b>	2	<b>n/a</b>	<b>10</b>	3	10	39	(2)
example 8	n/a	n/a	n/a	n/a	n/a	n/a	10	18	(1)
example 9	<b>n/a</b>	<b>3</b>	1	<b>n/a</b>	<b>4</b>	2	10	28	(2)
example 10	1	1	1	2	2	2	10	16	(1)
example 11	<b>2</b>	<b>10</b>	1	<b>n/a</b>	<b>15</b>	2	10	39	(2)
example 12	<b>n/a</b>	1	1	<b>n/a</b>	2	2	10	27	(1)
example 13	2	2	2	3	<b>5</b>	3	10	70	(2)(1)

# Evaluation: Experimental Study

- **Quantitative Comparison(2nd matched)**
  - In 8 examples **Strathcona** fails to find the 2nd match
  - 3 examples **MAPO** fails to find the 2nd match
  - **MAPO** requires to check fewer or the same number of snippets for the 2nd match than **Strathcona**.
  - In summary, **MAPO** requires less effort to search for rematched snippets than the other tools

**Table 2.** Comparison of Strathcona, Google code search, and MAPO

Example	First matched snippet			Second matched snippet			Total num. of items		
	Strat.	Google	MAPO	Strat.	Google	MAPO	Strat.	Google	MAPO
example 1	<b>5</b>	1	1	n/a	2	2	10	8	(2)
example 2	1	1	1	2	<b>n/a</b>	2	10	7	(1)
example 3	1	<b>3</b>	1	<b>5</b>	<b>4</b>	2	10	12	(4)(2)
example 4	n/a	4	n/a	n/a	10	n/a	10	11	n/a
example 5	1	<b>7</b>	2	3	<b>13</b>	3	10	33	(2)
example 6	n/a	9	n/a	n/a	11	n/a	10	33	n/a
example 7	2	<b>4</b>	2	<b>n/a</b>	<b>10</b>	3	10	39	(2)
example 8	n/a	n/a	n/a	n/a	n/a	n/a	10	18	(1)
example 9	<b>n/a</b>	<b>3</b>	1	<b>n/a</b>	<b>4</b>	2	10	28	(2)
example 10	1	1	1	2	2	2	10	16	(1)
example 11	<b>2</b>	<b>10</b>	1	<b>n/a</b>	<b>15</b>	2	10	39	(2)
example 12	<b>n/a</b>	1	1	<b>n/a</b>	2	2	10	27	(1)
example 13	2	2	2	3	<b>5</b>	3	10	70	(2)(1)

# Evaluation: Experimental Study

- **Impacts of MAPO's Design Decisions**

- Turn off MAPO's individual internal techniques
- **Selection**: helps MAPO mine frequent API method call sequences
- **Inlining**: helps MAPO mine API method calls from different methods of client code
- **Clustering**: helps MAPO alleviate the interlacement among different usages that are sensitive to programming tasks.

Table 3. Impacts of MAPO's design decisions

Example	First matched snippet				Second matched snippet				Total num. of items			
	All	×S	×I	×C	All	×S	×I	×C	All	×S	×I	×C
example 1	1	1	<b>n/a</b>	1	2	2	<b>n/a</b>	2	(2)	(2)	<b>n/a</b>	(2)
example 2	1	1	1	1	2	2	2	2	(1)	(1)	(1)	(1)
example 3	1	1	1	<b>n/a</b>	2	2	2	<b>n/a</b>	(4)(2)	(4)(2)	(3)(1)	<b>(2)</b>
example 4	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
example 5	2	2	2	<b>n/a</b>	3	3	3	<b>n/a</b>	(2)	(2)	(2)	<b>n/a</b>
example 6	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
example 7	2	2	2	<b>n/a</b>	3	3	3	<b>n/a</b>	(2)	(2)	(2)	<b>n/a</b>
example 8	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	(1)	(1)	(1)	(1)
example 9	1	1	1	1	2	2	2	2	(2)	(2)	(2)	(2)
example 10	1	1	1	1	2	2	2	2	(1)	(1)	(1)	(1)
example 11	1	1	1	<b>n/a</b>	2	2	2	<b>n/a</b>	(2)	(2)	(2)	<b>n/a</b>
example 12	1	1	1	1	2	2	2	2	(1)	(1)	(1)	(1)
example 13	2	<b>n/a</b>	<b>n/a</b>	<b>n/a</b>	3	<b>n/a</b>	<b>n/a</b>	<b>n/a</b>	(2)(1)	<b>(1)</b>	<b>(1)(1)</b>	<b>(1)</b>

In this table, we highlight those affected values with the bold font.

# Evaluation: Experimental Study

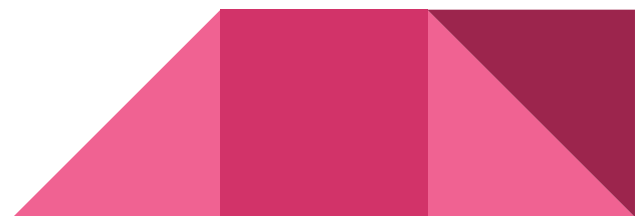
- **Impacts of MAPO's Design Decisions**

- **Clustering:** If  $|N|$  is small,  $s_1$ 's support value does not decrease much, and can still be a frequent sequence: 1, 2, 9, 10, and 12
- If  $|N|$  is large,  $s_1$ 's support value may decrease too much to be mined as a frequent sequence: 5, 7, and 11
- $N$  is the set of sequences that also call API methods in  $s_1$  from other clusters
- “# of API method call sequences with  $ss$ ”
- “# of API method call sequences in cluster  $C_1$ ”.

$$\frac{|With(s_1)|}{|C_1|} \rightarrow \frac{|With(s_1)|}{|C_1| + |N|}$$

ON

OFF



# Evaluation: Experimental Study

- **Threats to Validity**

- Only one set of APIs is used, and the recommendations are all on the use of GEF
- Code snippets are limited in number
- Code snippets from books may omit rare usages
- Some other code search engines or tools may perform better than these two tools



# Evaluation: Empirical Study

- Aims to investigate whether MAPO can assist programmers to complete programming tasks
- 6 programming tasks
- 6 graduate students (subjects) majoring in computer science from Peking University to complete the 6 tasks

**Table 4.** Tasks used in the empirical study

Task	Description	Essential API calls
1	Factor an incoming request	3
2	Start monitoring property changes	4
3	Update the name and the bounds of a figure	5
4	Add a context menu to an editor	5
5	Add a tool bar to an editor	5
6	Save the content of a editor	8

**Table 5.** Background of the subjects

	Group 1			Group 2		
	subject 1	subject 2	subject 3	subject 4	subject 5	subject 6
Java (Years)	4	3	2	3	1	3
GEF (Years)	2	0	0	0	0	1



# Evaluation: Empirical Study

- 2 stages
- In each stage, the 2 groups exchange their roles as the MAPO group and the control group
- 1st stage, completing task 1-3, Group 1 using Google code search and Strathcona, Group 2 using MAPO
- 2nd stage, completing task 4-6, Group 1 using MAPO

**Table 6.** Results of the empirical study

	Control Group				MAPO Group			
	<i>subject 1</i>	<i>subject 2</i>	<i>subject 3</i>	<i>total</i>	<i>subject 4</i>	<i>subject 5</i>	<i>subject 6</i>	<i>total</i>
Task 1	0	0	0	0	0	1	0	1
Task 2	0	1	1	2	0	1	1	2
Task 3	2	0	5	7	2	4	0	6
	MAPO Group				Control Group			
	<i>subject 1</i>	<i>subject 2</i>	<i>subject 3</i>	<i>total</i>	<i>subject 4</i>	<i>subject 5</i>	<i>subject 6</i>	<i>total</i>
Task 4	0	0	0	0	5	4	0	9
Task 5	0	0	0	0	0	4	0	4
Task 6	0	2	3	5	4	3	3	10

# Evaluation: Empirical Study

- In Tasks 1 and 2, there is a **little difference** between the MAPO group and the control group.
- In Task 3, there is also a **little difference** between the MAPO group and the control group.
- In Tasks 4, 5, and 6, there is a **significant difference** in performance between the MAPO group and the control group.
- In the task 4-6, as API usages are relatively complicated, MAPO successfully helps programmers produce code with fewer bugs than the other two tools

**Table 6.** Results of the empirical study

	Control Group				MAPO Group			
	<i>subject 1</i>	<i>subject 2</i>	<i>subject 3</i>	<i>total</i>	<i>subject 4</i>	<i>subject 5</i>	<i>subject 6</i>	<i>total</i>
Task 1	0	0	0	0	0	1	0	1
Task 2	0	1	1	2	0	1	1	2
Task 3	2	0	5	7	2	4	0	6
	MAPO Group				Control Group			
	<i>subject 1</i>	<i>subject 2</i>	<i>subject 3</i>	<i>total</i>	<i>subject 4</i>	<i>subject 5</i>	<i>subject 6</i>	<i>total</i>
Task 4	0	0	0	0	5	4	0	9
Task 5	0	0	0	0	0	4	0	4
Task 6	0	2	3	5	4	3	3	10

# Evaluation: Empirical Study

- **Threats to validity**

- Shares the threats with the study in **Experimental Study** as well
- Involves **human subjects**
- The results observed in the empirical study may not be applicable to programming tasks which are **not in the study**
- Control group can **use both** Google code search and Strathcona, which may have negative impacts on the two tools
- The **learning curve** of the these subjects may affect the results



# Conclusion

- MAPO: Help understand API usages and write API client code more effectively.
- MAPO implements a mechanism that combines frequent subsequence mining with clustering to mine API usage patterns from code snippets.
- MAPO provides a recommender that integrates with the existing Eclipse IDE. Through MAPO's recommender, a programmer can retrieve patterns to help navigate their associated snippets to find the code snippet of interest effectively.
- The study results show that MAPO helps a programmer to locate useful code snippets more effectively than existing tools Strathcona and Google code search



# Discussion

- Pros and cons on evaluation process
- Possible improvements on MAPO?
- Which tool/tools you will prefer to use?(MAPO, Google Code Search & Strathcona)



# References

- [1] M. Acharya, T. Xie, J. Pei, and J. Xu. Mining API patterns as partial orders from source code: From usage scenarios to specifications. In Proc. 7th ESEC/FSE, pages 25–34, 2007.
- [2] M. Aeschlimann, D. Baumer, and J. Lanneluc. Java tool smithing extending the Eclipse Java Development Tools. In Proc. 2nd EclipseCon, 2005.
- [3] R. Agrawal and R. Srikant. Mining sequential patterns. In Proc. 7th ICDE, pages 3–14, 1995.
- [4] R. Alur, P. Cerný, P. Madhusudan, and W. Nam. Synthesis of interface specifications for Java classes. In Proc. 32nd POPL, pages 98–109, 2005.
- [5] G. Ammons, R. Bodik, and J. R. Larus. Mining specifications. In Proc. 29th POPL, pages 4–16, 2002.
- [6] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.
- [7] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In Proc. 8th KDD, pages 429–435, 2002.
- [8] M. Bruch, T. Schäfer, and M. Mezini. FrUIT: IDE support for framework understanding. In Proc. 4th ETX, pages 55–59, 2006.

# References

- [9] R. Chang, A. Podgurski, and J. Yang. Finding what's not there: a new approach to revealing neglected conditions in software. In Proc. ISSTA, pages 163–173, 2007.
- [10] D. Engler, D. Y. Chen, S. Hallem, A. Chou, and B. Chelf. Bugs as deviant behavior: a general approach to inferring errors in systems code. In Proc. 8th SOSP, pages 57–72, 2001.
- [11] M. Gabel and Z. Su. Javert: fully automatic mining of general temporal properties from dynamic traces. In Proc. 16th FSE, pages 339–349, 2008.
- [12] Google Code Search Engine, 2008. <http://www.google.com/codesearch>.
- [13] J. Han and M. Kamber. Data mining: concepts and techniques. Morgan Kaufmann Publishers Inc., 2000.
- [14] T. Henzinger, R. Jhala, and R. Majumdar. Permissive interfaces. In Proc. 5th ESEC/FSE, pages 31–40, 2005.
- [15] R. Holmes and G. C. Murphy. Using structural context to recommend source code examples. In Proc. 27th ICSE, pages 117–125, 2005.
- [16] R. Holmes, R. J. Walker, and G. C. Murphy. Approximate structural context matching: An approach to recommend relevant examples. IEEE Transactions on Software Engineering, 32(12):952–970, 2006.

# References

- [17] R. Hudson and P. Shah. GEF in depth. In Proc. 2nd EclipseCon, 2005.
- [18] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.
- [19] Z. Li and Y. Zhou. PR-Miner: Automatically extracting implicit programming rules and detecting violations in large software code. In Proc. 5th ESEC/FSE, pages 306–315, 2005.
- [20] V. B. Livshits and T. Zimmermann. Dynamine: Finding common error patterns by mining software revision histories. In Proc. 5th ESEC/FSE, pages 296–305, 2005.
- [21] D. Lo and S. Khoo. SMaRTIC: towards building an accurate, robust and scalable specification miner. In Proc. 6th ESEC/FSE, pages 265–275, 2006.
- [22] D. Mandelin, L. Xu, R. Bodik, and D. Kimelman. Jungloid mining: helping to navigate the API jungle. In Proc. PLDI, pages 48–61, 2005.
- [23] S. N. Matthew Scarpino, Stephen Holder and L. Mihalkovic. *SWT/JFace in Action*. Manning, 2005.
- [24] F. McCarey, M. O. Cinnéide, and N. Kushmerick. Recommending library methods: An evaluation of the vector space model (VSM) and latent semantic indexing (LSI). In Proc. 9th ICSR, pages 217–230, 2006.



# References

- [25] A. Michail. Data mining library reuse patterns using generalized association rules. In Proc. 22nd ICSE, pages 167–176, 2000.
- [26] T. Ng, S. Cheung, W. Chan, and Y. Yu. Work experience versus refactoring to design patterns: a controlled experiment. In Proc. 6th ESEC/FSE, pages 12–22, 2006.
- [27] M. K. Ramanathan, A. Grama, and S. Jagannathan. Path-sensitive inference of function precedence protocols. In Proc. 29th ICSE, pages 240–250, 2007.
- [28] S. Reiss and M. Renieris. Encoding Program Executions. In Proc. 23rd ICSE, pages 221– 230, 2001.
- [29] Z. M. Saul, V. Filkov, P. Devanbu, and C. Bird. Recommending random walks. In Proc. 7th ESEC/FSE, pages 15–24, 2007.
- [30] C. Scaffidi. Why are APIs difficult to learn and use? Crossroads, 12(4):4–4, 2005.
- [31] S. Shoham, E. Yahav, S. Fink, and M. Pistoia. Static specification mining using automatabased abstractions. In Proc. ISSTA, pages 174–184, 2007.
- [32] N. Tansalarak and K. T. Claypool. XSnippet: Mining for sample code. In Proc. 21st OOPSLA, pages 413–430, 2006.

# References

- [33] S. Thummalapenta and T. Xie. PARSEWeb: A programmer assistant for reusing open source code on the web. In Proc. 22nd ASE, pages 204–213, 2007.
- [34] A. Wasylkowski, A. Zeller, and C. Lindig. Detecting object usage anomalies. In Proc. 7th ESEC/FSE, pages 35–44, 2007.
- [35] W. Weimer and G. Necula. Mining temporal specifications for error detection. In Proc. 11th TACAS, pages 461–476, 2005.
- [36] J. Whaley, M. Martin, and M. Lam. Automatic extraction of object-oriented component interfaces. In Proc. ISSTA, pages 218–228, 2002.
- [37] C. C. Williams and J. K. Hollingsworth. Recovering system specific rules from software repositories. In Proc. 2nd MSR, pages 1–5, 2005.
- [38] T. Xie and J. Pei. MAPO: Mining API usages from open source repositories. In Proc. 3rd MSR, pages 54–57, 2006.
- [39] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M. Das. Perracotta: mining temporal API rules from imperfect traces. In Proc. 28th ICSE, pages 282–291, 2006.

Thank You!!

