JavaParser: A Fine-Grain Concept Indexing Tool for Java Problems

Roya Hosseini, Peter Brusilovsky

Intelligent Systems Program (www.isp.pitt.edu)
University of Pittsburgh

First Workshop on AI-supported Education for Computer Science (AIEDCS)
Memphis, Tennessee, USA
July 13, 2013
Outline

• Introduction
• Hypothesis testing (pre-study)
• Findings and challenges
• JavaParser
• Conclusion & Future works
Introduction

There are two possible ways for modeling student’s knowledge:

• Coarse-grained knowledge modeling

• Fine-grained knowledge modeling
Motivation

Fine-grained indexing?

Do we really need it? ...
• An important aspect of task sequencing in Adaptive Hypermedia is the **granularity** of the domain model and the task indexing.

• In general, the sequencing algorithm can better determine the appropriate task if the granularity of the domain model and the task indexing is finer.

• However, fine-grained domain models that dissect a domain into dozens or hundreds of knowledge units are much harder to develop and to use for indexing.
Typical approach to present programming knowledge which uses coarse-grained topics like “loops” and “increment” allows reasonable sequencing during course. (Brusilovsky et al. 2009, Hsiao et al. 2010, Kavcic 2004, Vesin et al. 2012)

However, this approach fails in providing advance sequencing such as providing support for exam preparation or remediation.
Outline

• Introduction
• Hypothesis testing (pre-study)
• Findings and challenges
• JavaParser
• Conclusion & Future works
Pre-study

Knowledge Maximizer:

• An exam preparation tool for Java programming
• based on a fine-grained concept model of Java knowledge
• assumes a student already completed a considerable amount of work
• goal is to help her define gaps in knowledge and try to redress them as soon as possible.
```java
public class Tester {
    public static void main(String[] args) {
        Mechanism mech1 = new Computer(6.0, 2.0, true);
        Mechanism mech2 = new Car("Honda", 6);
        Computer comp = (Computer) mech1;
        System.out.println(comp.getProcessorSpeed());
        System.out.println(comp.reportProblems());
        System.out.println(((Car) mech2).getBrand());
        System.out.println(mech2.reportProblems());
    }
}
```

What is the output?
**KM Parameters:**

- *How prepared is the student to do the activity?*

\[
K = \frac{\sum_{i} k_{i}w'_{i}}{\sum_{i} w'_{i}}
\]

- **K**: user knowledge level
- **M_r**: prerequisite concepts
- **k_i**: knowledge in C_i
- **w_i**: weight of C_i for activity
• *How prepared is the student to do the activity?*

• *What is the impact of the activity?*

\[
I = \sum_{i}^{M_o} (1 - k_i)w'_i \quad \sum_{i}^{M_o} w'_i
\]

I : activity impact

\[M_o\] : outcome concepts
• How prepared is the student to do the activity?
• What is the impact of the activity?
• What is the value of repeating the activity again?

\[
\overline{S} = 1 - \frac{S}{t + 1}
\]

\(\overline{S}\) : inverse success rate for the activity
S : number of success in the activity
t : number of times the activity is done
• Determining the sequence of top 10 activity with the highest rank using:

\[ R = K + I + \overline{S} \]

- \( R \) : activity rank
- \( K \) : knowledge level in prerequisites of activity
- \( I \) : activity Impact
- \( \overline{S} \) : Inverse of success rate
Evaluation

• We conducted a classroom study for the Java Programming undergraduate course.

• Study started on Dec. 4th 2012 about a week before the final exam.

• The course also used QuizGuide (QG), and Progressor+ (P+) systems to access Java questions (available from the beginning of the semester).

• All these systems used same 103 parameterized Java questions.
Evaluation Measures

We grouped participants into two groups:

- KM: those who made at least ten attempt using KM (n = 9)
- QG/P+: those who made no attempt using KM and at least 10 attempt with QG/P+ (n = 16)

For each group we measured:

- Number of questions (attempts) done using each system
- Success Rate
Outline

• Introduction
• Hypothesis testing (pre-study)
• Findings and challenges
• JavaParser
• Conclusion & Future works
System Usage Summary

Attempts per Question Compelixty

- Easy: 6.20%
- Moderate: 43.50% to 45.30%
- Complex: 50.20% to 20.10%

AIEDCS 2013
Results

Success rate

<table>
<thead>
<tr>
<th>KM</th>
<th>QG,P+</th>
</tr>
</thead>
<tbody>
<tr>
<td>58%</td>
<td>64%</td>
</tr>
</tbody>
</table>

Average Relative Progress Percentage

<table>
<thead>
<tr>
<th>Class</th>
<th>QG/P+</th>
<th>KM</th>
</tr>
</thead>
<tbody>
<tr>
<td>-12.80%</td>
<td>-13.70%</td>
<td>11.53%</td>
</tr>
</tbody>
</table>
• Fine-grained knowledge modeling will push students toward appropriate complex questions which does not lead to miserable fails in those questions.

• During exam preparation, complex questions are more useful, since they target more concepts at once.

• This helps students fill the gaps in their knowledge in a more efficient way. (Eg. : 6 easy question must be done to get the same outcome as only 1 complex question!)
JAVA Ontology

http://www.sis.pitt.edu/~paws/ont/java.owl

~ 400 node in ontology
~ 160 concept (leaf nodes)
103 JAVA Parameterized Question

1-5 classes per question

5-52 concepts per question

~ 40% of questions has more than 20 concepts
52 concepts or more in a question?
JavaParser: A tool for automatic indexing of Java Problems
Outline

• Introduction
• Hypothesis testing (pre-study)
• Findings and challenges
• JavaParser
• Conclusion & Future works
Java Parser

• Developed using *Eclipse AST Tree* API
• The AST tree is semantically analyzed using the information in each of its nodes.
Structural properties of a method declaration

```java
public void start(BundleContext context) throws Exception {
    super.start(context);
}
```
Example

```java
public void start(BundleContext context) throws Exception {
    super.start(context);
}
```

Public Method Declaration
Example

```java
public void start(BundleContext context) throws Exception {
    super.start(context);
}
```

Return Type
Void
Example

```java
public void start(BundleContext context) throws Exception {
    super.start(context);
}
```

FormalMethodParameter
Single Variable Declaration
Example

```java
public void start(BundleContext context) throws Exception {
    super.start(context);
}
```

Exception
Example

```java
public void start(BundleContext context) throws Exception {
    super.start(context);
}
```

Super Method Invocation
• Current version of JavaParser is able to extract 98.77% of the concepts in 103 manually indexed questions.

• Average number of 8 extra concepts for each question indexed by automatic parsing.
Missed concept

- InheritanceBasedPolymorphism
- SuperclassSubclassConversion
- PolymorphicObjectCreationStatement
- MethodInheritance
- ....
Demo

http://adapt2.sis.pitt.edu/javaparser/ParseQuestion.jsp
Outline

• Introduction
• Hypothesis testing (pre-study)
• Findings and challenges
• JavaParser

• Conclusion & Future works
Future work

Use the results of fine-indexing for:

• Improving the user modeling service
• Cross content sequencing and providing remediation in case of failing a question.
• Predicting parts of code that might lead to student failure and provide hints accordingly.
• Expand the parser to extract more elaborated concepts and programming patterns.
Thank you!

Contact: roh38@pitt.edu