Speculative Analysis of IDE Recommendations

Kıvanç Muşlu, Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin

University of Washington
University of Massachusetts Amherst
University of Waterloo
IDE Recommendations

• aim to increase developer speed & confidence
• are widely used by developers

[Murphy et al. 2006]
Making recommendations more useful

Present
- IDE generates the recommendations
- Developer selects a recommendation based on experience

Today
- IDE generates recommendations & computes their consequences
- Developer selects a **better** recommendation **faster**
1 Logical Problem but 2 Errors

```java
public class QFS_Demo {
    private String name;
    public void setArg(String name) {
        name = name;
    }
}
```
Proposals at declaration can be prioritized better
Proposals at assignment do not help
Ultimate Goal

Best fix from any location
Consequences of IDE Recommendations

Problem: IDEs do not show the consequences of each recommendation

Solution: Computing and showing the consequences can increase developer productivity
Outline

• Motivation
• Quick Fix Scout (Speculative Analysis)
• Demo
• Evaluation
• Related Work
• Contributions
Running Speculative Analysis

```java
public class QFS_Demo {
    private String name_;  
    public void setArg(String name) {
        name_ = name;
    }
}
```
Running Speculative Analysis

Create class 'string'

```java
public class QFS_Demo {

    private string name;

    public void setArg(String name) {
        name_ = name;
    }

    class string {}
}
```

Create interface 'string'

```java
public class QFS_Demo {

    private string name;

    public void setArg(String name) {
        name_ = name;
    }

    interface string {}
}
```

Change to 'Spring' (javax.swing)

```java
public class QFS_Demo {

    private String name;

    public void setArg(String name) {
        name_ = name;
    }

    import javax.swing.Spring;
}
```

Change to 'String' (java.lang)

```java
public class QFS_Demo {

    private String name;

    public void setArg(String name) {
        name_ = name;
    }
}
```
Augmented Dialog with Speculative Compilation Error Counts
Making Quick Fix Global

```java
public class QFS_Demo {
    private String name_; 
    
    public void setArg(String name) {
        name_ = name;
    }
}
```

---

(0) Change to 'String' (java.lang)

```java
public class QFS_Demo {
    private String name_; 
    
    public void setArg(String name) {
        name_ = name;
    }
}
```

---

(2) Create class 'name_'
(2) Create interface 'name_'
(2) Change to 'Name' (javax.lang.model.element)
(2) Change to 'Name' (javax.naming)
(2) Change to 'Name' (javax.xml.soap)
(2) Change to 'NameList' (org.w3c.dom)
(2) Change to 'Naming' (java.rmi)
(2) Create enum 'name_
(2) Add type parameter 'name_' to 'QFS_Demo'
(2) Add type parameter 'name_' to 'setArg(String)'
(2) Fix project setup...
(2) Change to 'Name' (java.util.jar.Attributes)

Press '§1' to go to original position
Global Best Proposal

```java
public class QFS_Demo {

    private String name_;

    public void setArg(String name) {
        name_ = name;
    }
}
```

- (0) QFS_Demo.java:4:12: Change 'string' to 'String' (java.lang)
- Create class 'name_'
- Create interface 'name_'
- Change to 'Name' (javax.lang.model.element)
- Change to 'Name' (javax.naming)
- Change to 'Name' (javax.xml.soap)
- Change to 'NameList' (org.w3c.dom)
- Change to 'Naming' (java.rmi)
- Create enum 'name_'
- Add type parameter 'name_' to 'QFS_Demo'
- Add type parameter 'name_' to 'setArg(String)'
- Fix project setup...
- Change to 'Name' (java.util.jar.Attributes)

Press '^1' to go to original position
Evaluation

• Controlled experiment of Quick Fix Scout
  – 20 grad students

• Case study with 13 participants on how developers use Quick Fix
  – Details presented in the paper
Controlled Experiment

RQ1: Does QFS speed up fixing compilation errors?

RQ2: Does QFS change developer behavior?

- 24 project snapshots with compilation errors
  - Chosen randomly from the case study participants’ development history
  - Mutation compilation errors were added to half of the tasks
  - Within-participant mixed design, 2 factors: tool & ordering
Controlled Experiment Results

Proposal Selection
• Best Proposal selected 87% with QFS, 73% without it
• Global Best Proposal selected 75% when offered

Bug Removal Time
• Better by 12% (3 minutes)

Quick Fix Dialog Invocations
• Users spent 0.8 seconds (22%) more examining QFS dialogs

➡️ Without QFS users needed more manual exploration
➡️ QFS provides users more relevant information
Participant Quotations

“I could tell [Quick Fix Scout] wasn’t just saving me time, but increasing my understanding of the program.”

“Where can I use [Quick Fix Scout] in my own Eclipse?...Debugging with [Quick Fix Scout] felt much faster and less stressful.”
Related Work

• Improving existing recommendations
  – Historical information & heuristics
    [Robbes et al. 2008] [Bruch et al. 2009]
    QFS computes consequences precisely

• Defining new recommendations
  [Castro-Herrera et al. 2009] [Xiang et al. 2008]
  – Using extra type information to chain API calls
    [Perelman et al. 2012]
    QFS analyzes existing recommendations
    QFS can exploit these new recommendations
Contributions

• Speculation for IDE recommendations
• Implementation: Quick Fix Scout
  http://quick-fix-scout.googlecode.com
• Preliminary evidence of usefulness
References


Towards fine-grained speculation

- QFS always maintains a copy project that is in sync with the original one
- The speculative analysis starts as soon as a change in compilation errors is detected
- QFS caches the compilation errors
- QFS caches proposals
- QFS is aware of the active file and cursor location, the compilation errors are prioritized accordingly.
## 6 Popular Proposals

<table>
<thead>
<tr>
<th>Proposal Name</th>
<th>Selection Rate</th>
<th>Top 3 Offer Frequency (Selected)</th>
<th>Top 3 Offer Frequency (All)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import <code>&lt;type name&gt;</code></td>
<td>24%</td>
<td>12%</td>
<td>12%</td>
</tr>
<tr>
<td>Add throws declaration</td>
<td>23%</td>
<td>9%</td>
<td>6%</td>
</tr>
<tr>
<td>Create method <code>&lt;method name&gt;</code></td>
<td>15%</td>
<td>6%</td>
<td>5%</td>
</tr>
<tr>
<td>Change to <code>&lt;new name&gt;</code></td>
<td>9%</td>
<td>8%</td>
<td>11%</td>
</tr>
<tr>
<td>Add unimplemented methods</td>
<td>7%</td>
<td>3%</td>
<td>2%</td>
</tr>
<tr>
<td>Surround with try/catch</td>
<td>4%</td>
<td>9%</td>
<td>6%</td>
</tr>
<tr>
<td><strong>All</strong></td>
<td><strong>82%</strong></td>
<td><strong>47%</strong></td>
<td><strong>42%</strong></td>
</tr>
<tr>
<td>Create class <code>&lt;type name&gt;</code></td>
<td>~0%</td>
<td>8%</td>
<td>9%</td>
</tr>
<tr>
<td>Create interface <code>&lt;type name&gt;</code></td>
<td>~0%</td>
<td>4%</td>
<td>5%</td>
</tr>
</tbody>
</table>
## 6 Popular Proposals (Per User)

<table>
<thead>
<tr>
<th>Proposal Name</th>
<th>U01</th>
<th>U02</th>
<th>U03</th>
<th>U04</th>
<th>U05</th>
<th>U06</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import <code>&lt;type name&gt;</code></td>
<td>24%</td>
<td>2%</td>
<td>76%</td>
<td>34%</td>
<td>37%</td>
<td>53%</td>
<td>24%</td>
</tr>
<tr>
<td>Add throws declaration</td>
<td>21%</td>
<td>47%</td>
<td>0%</td>
<td>11%</td>
<td>0%</td>
<td>18%</td>
<td>23%</td>
</tr>
<tr>
<td>Create method <code>&lt;method name&gt;</code></td>
<td>21%</td>
<td>11%</td>
<td>0%</td>
<td>2%</td>
<td>11%</td>
<td>0%</td>
<td>15%</td>
</tr>
<tr>
<td>Change to <code>&lt;new name&gt;</code></td>
<td>10%</td>
<td>1%</td>
<td>6%</td>
<td>26%</td>
<td>7%</td>
<td>24%</td>
<td>9%</td>
</tr>
<tr>
<td>Add unimplemented methods</td>
<td>7%</td>
<td>8%</td>
<td>0%</td>
<td>0%</td>
<td>9%</td>
<td>6%</td>
<td>7%</td>
</tr>
<tr>
<td>Surround with try/catch</td>
<td>0%</td>
<td>14%</td>
<td>0%</td>
<td>6%</td>
<td>0%</td>
<td>0%</td>
<td>4%</td>
</tr>
</tbody>
</table>
Cluttering of Workspace

• There are many filters & workaround to reduce cluttering
  – Quick Fix Scout creates a working set called ‘QFS’ and puts all copy projects under this working set
  – It updates some of the filters (navigation, package manager) automatically when installed to hide copy project.
  – Users can manually update some settings to reduce cluttering

• With Eclipse 4, Eclipse might be able to run multiple workspaces (Quick Fix Scout can create a private workspace)
Limitations

• Quick Fix through Hover Dialog does not work
  – Hover dialog uses a different API to create proposals and Eclipse does not permit us to override that code

• For interactive proposals (Create class/enum, etc.) we cannot compute the remaining errors

• For two proposals (Change type name and change compilation unit name), we cannot compute the remaining errors due to a bug in their implementation
  – Undo changes are implemented incorrectly
Quick Fix Scout Algorithm

```java
while (true) {
    for (Error error: project.getErrors()) {
        for (Proposal prop: error.getProposals()) {
            copy.applyProposal(prop);
            result.put(prop, copy.getErrors());
            copy.applyProposal(prop.getUndo());
        }
    }
}
```