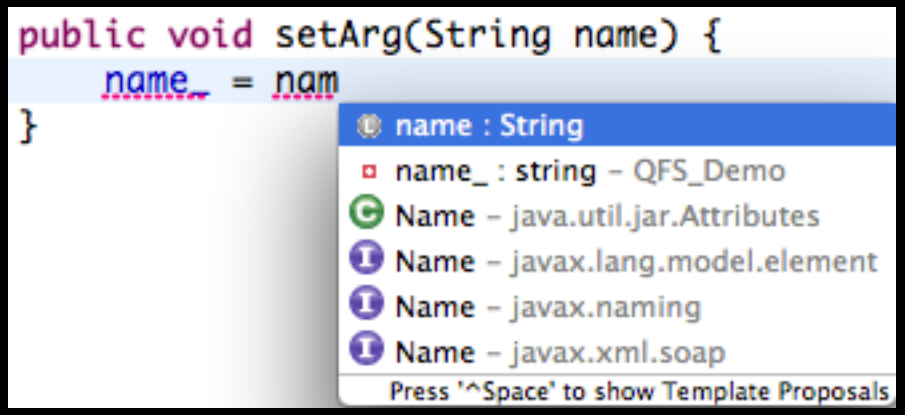
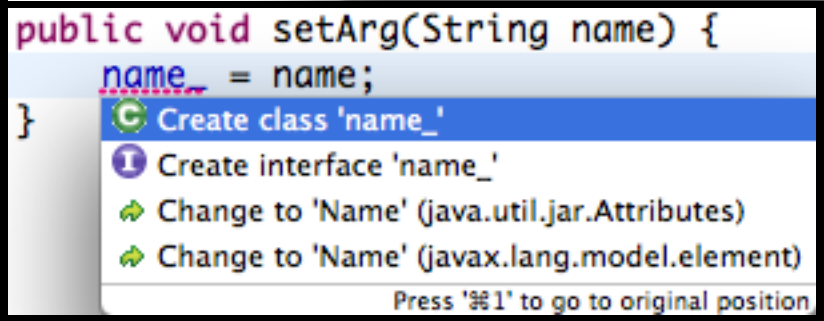
Speculative Analysis of IDE Recommendations

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

```
public void setArg(String name) {
    name_ = name;
}
    (0) QFS_Demo.java:4:12: Change 'string' to 'String' (java.lang)
    (2) Create class 'name_'
    (2) Create interface 'name_'
    (2) Change to 'Name' (javax.lang.model.element)
                                Press '⌘1' to go to original position
```

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

```
public class QFS_Demo {

    private string name_;

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

Logical Problem

Compilation error 1

Compilation error 2

Proposals at declaration can be prioritized better

```
public class QFS_Demo {

    private string name_;

    public v
    {
        name
    }
}
```

- Create class 'string'
- Create interface 'string'
- Change to 'Spring' (javax.swing)
- Change to 'String' (java.lang)
- Change to 'STRING' (javax.print.DocFlavor)
- Change to 'StringBuffer' (java.lang)
- Change to 'StringHolder' (org.omg.CORBA)
- Change to 'StringReader' (java.io)
- Change to 'StringWriter' (java.io)
- Create enum 'string'
- Add type parameter 'string' to 'QFS_Demo'
- Fix project setup...

Press '⌘1' to go to original position

Proposals at assignment do not help

```
public class QFS_Demo {

    private string name_;

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

- Create class 'name_'
- Create interface 'name_'
- Change to 'Name' (java.util.jar.Attributes)
- 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'
- Rename in file (⌘2 R)
- Rename in workspace (⌥⌘R)
- Add type parameter 'name_' to 'setArg(String)'
- Fix project setup...

Ultimate Goal

```
public class QFS_Demo {

    private string name_;

    public v
    {
        name
    }
}
```

Best fix

- Change to 'String' (java.lang)
- Create class 'string'
- Create interface 'string'
- Change to 'STRING' (javax.print.DocFlavor)
- Change to 'Spring' (javax.swing)
- Change to 'StringBuffer' (java.lang)
- Change to 'StringHolder' (org.omg.CORBA)
- Change to 'StringReader' (java.io)
- Change to 'StringWriter' (java.io)
- Create enum 'string'
- Add type parameter 'string' to 'QFS_Demo'
- Fix project setup...

Press '⌘1' to go to original position

```
public class QFS_Demo {

    private string nar

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

Best fix from any location

- Change 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

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

```
public class QFS_Demo {

    private string name_;

    public v
    {
        name
    }
}
```

Create class 'string'
Create interface 'string'
Change to 'Spring' (javax.swi
Change to 'String' (java.lang)

```
public class QFS_Demo {

    private string name_;

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

```
public class QFS_Demo {

    private string name_;

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

```
public class QFS_Demo {

    private string name_;

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

Running Speculative Analysis

Create class 'string'

```
public class QFS_Demo {

    private string name_;

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

1

Create interface 'string'

```
public class QFS_Demo {

    private string name_;

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

1

Change to 'Spring' (javax.swing)

```
public class QFS_Demo {

    private Spring name_;

    public void setArg(String name)
    {
        name_ = name;
    }
}   import javax.swing.Spring;
```

1

Change to 'String' (java.lang)

```
public class QFS_Demo {

    private String name_;

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

0

Augmented Dialog with Speculative Compilation Error Counts

```
public class QFS_Demo {

    private string name_;

    public v
    {
        name
    }
}
```

- (0) Change to 'String' (java.lang)
- (1) Create class 'string'
- (1) Create interface 'string'
- (1) Change to 'STRING' (javax.print.DocFlavor)
- (1) Change to 'Spring' (javax.swing)
- (1) Change to 'StringBuffer' (java.lang)
- (1) Change to 'StringHolder' (org.omg.CORBA)
- (1) Change to 'StringReader' (java.io)
- (1) Change to 'StringWriter' (java.io)
- (1) Create enum 'string'
- (1) Add type parameter 'string' to 'QFS_Demo'
- (2) Fix project setup...

Press '⌘1' to go to original position

Making Quick Fix Global

```
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

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

```
public class QFS_Demo {
    private String name_;

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

Global Best Proposal

```
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)
- (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

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

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

(0) QFS_Demo.java:4:12: Change 'string' to 'String' (java.lang)
(2) Create class 'name_'
(2) Create interface 'name_'
(2) Change to 'Name' (javax.lang.model.element)

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

References

M. Bruch, M. Monperrus, and M. Mezini. Learning from examples to improve code completion systems. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering,* pages 213–222, Amsterdam, The Netherlands, 2009. doi: 10.1145/1595696.1595728.

Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin. Speculative analysis: Exploring future states of software. In *Proceedings of the 2010 Foundations of Software Engineering Working Conference on the Future of Software Engineering Research (FoSER10),* Santa Fe, NM, USA, November 2010.

Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin. Crystal: Proactive conflict detector for distributed version control. http://crystalvc.googlecode.com, 2010.

Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin. Proactive detection of collaboration conflicts. In *Proceedings of the 8th Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE11),* pages 168–178, Szeged, Hungary, September 2011. doi: 10.1145/ 2025113.2025139.

C. Castro-Herrera, C. Duan, J. Cleland-Huang, and B. Mobasher. A recommender system for requirements elicitation in large-scale software projects. In *Proceedings of the 2009 ACM symposium on Applied Computing,* SAC ’09, pages 1419–1426, 2009. doi: 10.1145/1529282.1529601.

K. Muşlu, Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin. Improving IDE Recommendations by Considering Global Implications of Existing Recommendations. In *Proceedings of the 34th International Conference on Software Engineering, New Ideas and Emerging Results Track,* ICSE ’12, Zurich, Switzerland, June 2012.

G. C. Murphy, M. Kersten, and L. Findlater. How are java software developers using the eclipse ide? *IEEE Software,* 23 (4):76–83, July 2006. doi: 10.1109/MS.2006.105.

P. F. Xiang, A. T. T. Ying, P. Cheng, Y. B. Dang, K. Ehrlich, M. E. Helander, P. M. Matchen, A. Empere, P. L. Tarr, C. Williams, and S. X. Yang. Ensemble: a recommendation tool for promoting communication in software teams. In *Proceedings of the 2008 international workshop on Recommendation systems for software engineering,* RSSE ’08, pages 2:1–2:1, 2008. doi: 10.1145/1454247.1454259.

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

Proposal Name	Selection Rate	Top 3 Offer Frequency (Selected)	Top 3 Offer Frequency (All)
Import <type name>	24%	12%	12%
Add throws declaration	23%	9%	6%
Create method <method name>	15%	6%	5%
Change to <new name>	9%	8%	11%
Add unimplemented methods	7%	3%	2%
Surround with try/catch	4%	9%	6%
All	**82%**	**47%**	**42%**
Create class <type name>	~0%	8%	9%
Create interface <type name>	~0%	4%	5%

6 Popular Proposals (Per User)

Proposal Name	U01	U02	U03	U04	U05	U06	All
Import <type name>	24%	2%	76%	34%	37%	53%	**24%**
Add throws declaration	21%	47%	0%	11%	0%	18%	**23%**
Create method <method name>	21%	11%	0%	2%	11%	0%	**15%**
Change to <new name>	10%	1%	6%	26%	7%	24%	**9%**
Add unimplemented methods	7%	8%	0%	0%	9%	6%	**7%**
Surround with try/catch	0%	14%	0%	6%	0%	0%	**4%**

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

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