

Comparing Fine-Grained Source Code Changes And Code Churn For Bug Prediction

Working Conference on Mining Software Repositories 2011

Emanuel Giger¹, Martin Pinzger², Harald Gall¹

¹University of Zurich, Switzerland

²Delft University of Technology, The Netherlands



University of Zurich
Department of Informatics



Bug Prediction

- Many useful papers on building bug prediction models
- Product measures, process measures, organizational measures - or a combination
- Process measures performed particularly well
- Very popular: Revisions and Code Churn

Revisions are coarse grained

There is more than just a *file revision*

```
private IStructureComparator fStructureComparator;

public boolean setInput(ITypedElement newInput, boolean force) {
    boolean changed = false;
    if (force || newInput != fInput) {
        removeDocumentRangeUpdaters();
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangeListener);
        fInput = newInput;
        if (fInput == null) {
            if (fStructureComparator instanceof IDisposable) {
                IDisposable disposable = (IDisposable) fStructureComparator;
                disposable.dispose();
            }
            fStructureComparator = null;
        } else {
            refresh();
            changed = true;
        }
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangeListener);
    }
    return changed;
}
```

```
/**
 * Remove any document range updaters that were registered against the document.
 */
private void removeDocumentRangeUpdaters() {
    if (fStructureComparator instanceof IDocumentRange) {
        IDocument doc = ((IDocumentRange) fStructureComparator).getDocument();
        try {
            // ...
        }
    }
}
```

```
private ITypedElement fInput;
private IStructureComparator fStructureComparator;

public boolean setInput(ITypedElement newInput, boolean force) {
    boolean changed = false;
    if (force || newInput != fInput) {
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangeListener);
        fInput = newInput;
        if (fInput != null) {
            refresh();
            changed = true;
        } else {
            if (fStructureComparator instanceof IDisposable) {
                IDisposable disposable = (IDisposable) fStructureComparator;
                disposable.dispose();
            }
            fStructureComparator = null;
        }
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangeListener);
    }
    return changed;
}
```

```
public IStructureComparator getStructureComparator() {
    return fStructureComparator;
}

public void refresh() {
    IStructureComparator oldComparator = fStructureComparator;
    fStructureComparator = createStructure();
    // Dispose of the old one after it has been used in a shared document
}
```

Revisions are coarse grained

There is more than just a *file revision*

```
private IStructureComparator fStructureComparator;

public boolean setInput(ITypedElement newInput, boolean force) {
    boolean changed = false;
    if (force || newInput != fInput) {
        removeDocumentRangeUpdaters();
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangeListener);
        fInput = newInput;
        if (fInput == null) {
            if (fStructureComparator instanceof IDisposable) {
                IDisposable disposable = (IDisposable) fStructureComparator;
                disposable.dispose();
            }
            fStructureComparator = null;
        } else {
            refresh();
            changed = true;
        }
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangeListener);
    }
    return changed;
}
```

removeDocumentRangeUpdaters();

```
/**
 * Remove any document range updaters that were registered against the document.
 */
private void removeDocumentRangeUpdaters() {
    if (fStructureComparator instanceof IDocumentRange) {
        IDocument doc = ((IDocumentRange) fStructureComparator).getDocument();
        try {
            doc.removeRangeUpdaters(fStructureComparator);
        } catch (Exception e) {
            // ignore
        }
    }
}
```

```
private ITypedElement fInput;
private IStructureComparator fStructureComparator;

public boolean setInput(ITypedElement newInput, boolean force) {
    boolean changed = false;
    if (force || newInput != fInput) {
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangeListener);
        fInput = newInput;
        if (fInput != null) {
            refresh();
            if (fStructureComparator instanceof IDisposable) {
                IDisposable disposable = (IDisposable) fStructureComparator;
                disposable.dispose();
            }
            fStructureComparator = null;
        }
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangeListener);
    }
    return changed;
}
```

```
public IStructureComparator getStructureComparator() {
    return fStructureComparator;
}

public void refresh() {
    IStructureComparator oldComparator = fStructureComparator;
    fStructureComparator = createStructure();
    // Dispose of the old one after it has been used in a shared document
}
```

Revisions are coarse grained

There is more than just a *file revision*

```
private IStructureComparator fStructureComparator;

public boolean setInput(ITypedElement newInput, boolean force) {
    boolean changed = false;
    if (force || newInput != fInput) {
        removeDocumentRangeUpdaters();
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangeListener);
        fInput = newInput;
        if (fInput == null) {
            if (fStructureComparator instanceof IDisposable) {
                IDisposable disposable = (IDisposable) fStructureComparator;
                disposable.dispose();
            }
            fStructureComparator = null;
        } else {
            refresh();
            changed = true;
        }
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangeListener);
    }
    return changed;
}
```

```
/**
 * Remove any document range updaters that were registered against the document.
 */
private void removeDocumentRangeUpdaters() {
    if (fStructureComparator instanceof IDocumentRange) {
        IDocument doc = ((IDocumentRange) fStructureComparator).getDocument();
        try {
            // ...
        }
    }
}
```

```
private ITypedElement fInput;
private IStructureComparator fStructureComparator;

public boolean setInput(ITypedElement newInput, boolean force) {
    boolean changed = false;
    if (force || newInput != fInput) {
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangeListener);
        fInput = newInput;
        if (fInput != null) {
            refresh();
            changed = true;
        } else {
            if (fStructureComparator instanceof IDisposable) {
                IDisposable disposable = (IDisposable) fStructureComparator;
                disposable.dispose();
            }
            fStructureComparator = null;
        }
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangeListener);
    }
    return changed;
}
```

```
public IStructureComparator getStructureComparator() {
    return fStructureComparator;
}

public void refresh() {
    IStructureComparator oldComparator = fStructureComparator;
    fStructureComparator = createStructure();
    // Dispose of the old one after it has been used in a shared document
}
```

Revisions are coarse grained

There is more than just a *file revision*

```
private IStructureComparator fStructureComparator;

public boolean setInput(ITypedElement newInput, boolean force) {
    boolean changed = false;
    if (force || newInput != fInput) {
        removeDocumentRangeUpdaters();
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangeListener);
        fInput = newInput;
        if (fInput == null) {
            if (fStructureComparator instanceof IDisposable) {
                IDisposable disposable = (IDisposable) fStructureComparator;
                disposable.dispose();
            }
            fStructureComparator = null;
        } else {
            refresh();
            changed = true;
        }
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangeListener);
    }
    return changed;
}
```

```
/**
 * Remove any document range updaters that were registered against the document
 */
private void removeDocumentRangeUpdaters() {
    if (fStructureComparator instanceof IDocumentRange) {
        IDocument doc = ((IDocumentRange) fStructureComparator).getDocument();
        try {
            doc.removeRangeUpdaters(fStructureComparator);
        } catch (Exception e) {
            // ignore
        }
    }
}
```

```
fInput = newInput;
if (fInput == null) {
    if (fStructureComparator instanceof IDisposable) {
        IDisposable disposable = (IDisposable) fStructureComparator;
        disposable.dispose();
    }
    fStructureComparator = null;
} else {
    refresh();
    changed = true;
}
```

```
private ITypedElement fInput;
private IStructureComparator fStructureComparator;

public boolean setInput(ITypedElement newInput, boolean force) {
    boolean changed = false;
    if (force || newInput != fInput) {
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangeListener);
        fInput = newInput;
        if (fInput != null) {
            refresh();
            changed = true;
        } else {
            if (fStructureComparator instanceof IDisposable) {
                IDisposable disposable = (IDisposable) fStructureComparator;
                disposable.dispose();
            }
            fStructureComparator = null;
        }
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangeListener);
    }
    return changed;
}
```

```
IStructureComparator getStructureComparator() {
    return fStructureComparator;
}

public void refresh() {
    IStructureComparator oldComparator = fStructureComparator;
    fStructureComparator = createStructure();
    // Dispose of the old one after it has been replaced by the new one
    if (oldComparator instanceof IDisposable)
        oldComparator.dispose();
}
```

Revisions are coarse grained

There is more than just a *file revision*

```
private IStructureComparator fStructureComparator;

public boolean setInput(ITypedElement newInput, boolean force) {
    boolean changed = false;
    if (force || newInput != fInput) {
        removeDocumentRangeUpdaters();
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangeListener);
        fInput = newInput;
        if (fInput == null) {
            if (fStructureComparator instanceof IDisposable) {
                IDisposable disposable = (IDisposable) fStructureComparator;
                disposable.dispose();
            }
            fStructureComparator = null;
        } else {
            refresh();
            changed = true;
        }
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangeListener);
    }
    return changed;
}
```

```
/**
 * Remove any document range updaters that were registered against the document.
 */
private void removeDocumentRangeUpdaters() {
    if (fStructureComparator instanceof IDocumentRange) {
        IDocument doc = ((IDocumentRange) fStructureComparator).getDocument();
        try {
            // ...
        }
    }
}
```

```
private ITypedElement fInput;
private IStructureComparator fStructureComparator;

public boolean setInput(ITypedElement newInput, boolean force) {
    boolean changed = false;
    if (force || newInput != fInput) {
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangeListener);
        fInput = newInput;
        if (fInput != null) {
            refresh();
            changed = true;
        } else {
            if (fStructureComparator instanceof IDisposable) {
                IDisposable disposable = (IDisposable) fStructureComparator;
                disposable.dispose();
            }
            fStructureComparator = null;
        }
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangeListener);
    }
    return changed;
}
```

```
public IStructureComparator getStructureComparator() {
    return fStructureComparator;
}

public void refresh() {
    IStructureComparator oldComparator = fStructureComparator;
    fStructureComparator = createStructure();
    // Dispose of the old one after it has been used in a shared document
}
```

Revisions are coarse grained

There is more than just a *file revision*

```
private IStructureComparator fStructureComparator;

public boolean setInput(ITypedElement newInput, boolean force) {
    boolean changed = false;
    if (force || newInput != fInput) {
        removeDocumentRangeUpdaters();
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangeListener);
        fInput = newInput;
        if (fInput == null) {
            if (fStructureComparator instanceof IDisposable) {
                IDisposable disposable = (IDisposable) fStructureComparator;
                disposable.dispose();
            }
            fStructureComparator = null;
        } else {
            refresh();
            changed = true;
        }
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangeListener);
    }
    return changed;
}
```

```
/**
 * Remove any document range updaters that were registered against the document.
 */
```

```
private void removeDocumentRangeUpdaters() {
    if (fStructureComparator instanceof IDocumentRange) {
        IDocumentRange docRange = (IDocumentRange) fStructureComparator;
        IDocument doc = docRange.getDocument();
        try {
            doc.removeRangeUpdater(docRange);
        } catch (Exception e) {
            // ignore
        }
    }
}
```

```
private void removeDocumentRangeUpdaters() {
    if (fStructureComparator instanceof IDocumentRange) {
        IDocument doc = ((IDocumentRange) fStructureComparator).getDocument();
        try {
            doc.removeRangeUpdater(fStructureComparator);
        } catch (Exception e) {
            // ignore
        }
    }
}
```

```
private ITypedElement fInput;
private IStructureComparator fStructureComparator;

public boolean setInput(ITypedElement newInput, boolean force) {
    boolean changed = false;
    if (force || newInput != fInput) {
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangeListener);
        fInput = newInput;
        if (fInput != null) {
            refresh();
            changed = true;
        } else {
            if (fStructureComparator instanceof IDisposable) {
                IDisposable disposable = (IDisposable) fStructureComparator;
                disposable.dispose();
            }
            fStructureComparator = null;
        }
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangeListener);
    }
    return changed;
}
```

```
public IStructureComparator getStructureComparator() {
    return fStructureComparator;
}
```

Revisions are coarse grained

There is more than just a *file revision*

```
private IStructureComparator fStructureComparator;

public boolean setInput(ITypedElement newInput, boolean force) {
    boolean changed = false;
    if (force || newInput != fInput) {
        removeDocumentRangeUpdaters();
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangeListener);
        fInput = newInput;
        if (fInput == null) {
            if (fStructureComparator instanceof IDisposable) {
                IDisposable disposable = (IDisposable) fStructureComparator;
                disposable.dispose();
            }
            fStructureComparator = null;
        } else {
            refresh();
            changed = true;
        }
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangeListener);
    }
    return changed;
}
```

```
/**
 * Remove any document range updaters that were registered against the document.
 */
private void removeDocumentRangeUpdaters() {
    if (fStructureComparator instanceof IDocumentRange) {
        IDocument doc = ((IDocumentRange) fStructureComparator).getDocument();
        try {
            // ...
        }
    }
}
```

```
private ITypedElement fInput;
private IStructureComparator fStructureComparator;

public boolean setInput(ITypedElement newInput, boolean force) {
    boolean changed = false;
    if (force || newInput != fInput) {
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangeListener);
        fInput = newInput;
        if (fInput != null) {
            refresh();
            changed = true;
        } else {
            if (fStructureComparator instanceof IDisposable) {
                IDisposable disposable = (IDisposable) fStructureComparator;
                disposable.dispose();
            }
            fStructureComparator = null;
        }
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangeListener);
    }
    return changed;
}
```

```
public IStructureComparator getStructureComparator() {
    return fStructureComparator;
}

public void refresh() {
    IStructureComparator oldComparator = fStructureComparator;
    fStructureComparator = createStructure();
    // Dispose of the old one after it has been used in a shared document
}
```

Code Churn can be imprecise

Regarding the type and the semantics of source code changes

```
/* Copyright (c) 2000, 2004 IBM Corporation and others.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 *
 * Contributors:
 *   IBM Corporation - initial API and implementation
 */
package org.eclipse.compare.structuremergeviewer;

import org.eclipse.swt.events.DisposeEvent;
import org.eclipse.swt.widgets.*;
import org.eclipse.jface.util.PropertyChangeEvent;

import org.eclipse.compare.*;
import org.eclipse.compare.internal.*;

/**
```

```
/* Copyright (c) 2000, 2004 IBM Corporation and others.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Common Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/cpl-v10.html
 *
 * Contributors:
 *   IBM Corporation - initial API and implementation
 */
package org.eclipse.compare.structuremergeviewer;

import org.eclipse.swt.events.DisposeEvent;
import org.eclipse.swt.widgets.*;
import org.eclipse.jface.util.PropertyChangeEvent;

import org.eclipse.compare.*;
import org.eclipse.compare.internal.*;

/**
```

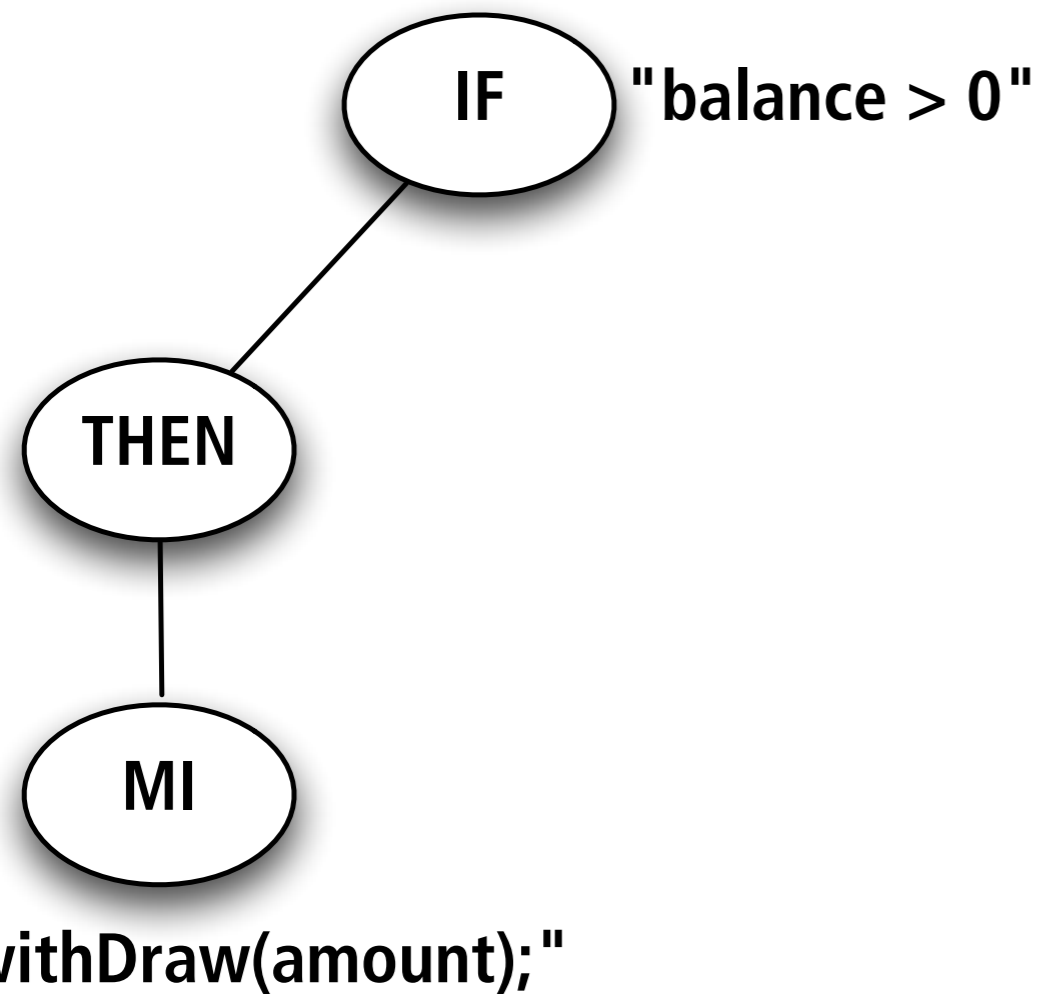
Fine Grained-Source Code Changes (SCC)

- SCC leverage the implicit code structure of the abstract syntax tree (AST)
- SCC are extracted using a tree differencing algorithm that compares the ASTs of two revisions of a file¹

¹Beat Fluri, Michael Würsch, Martin Pinzger, Harald C. Gall, **Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction**, *IEEE Transactions on Software Engineering* Vol. 33 (11), November 2007

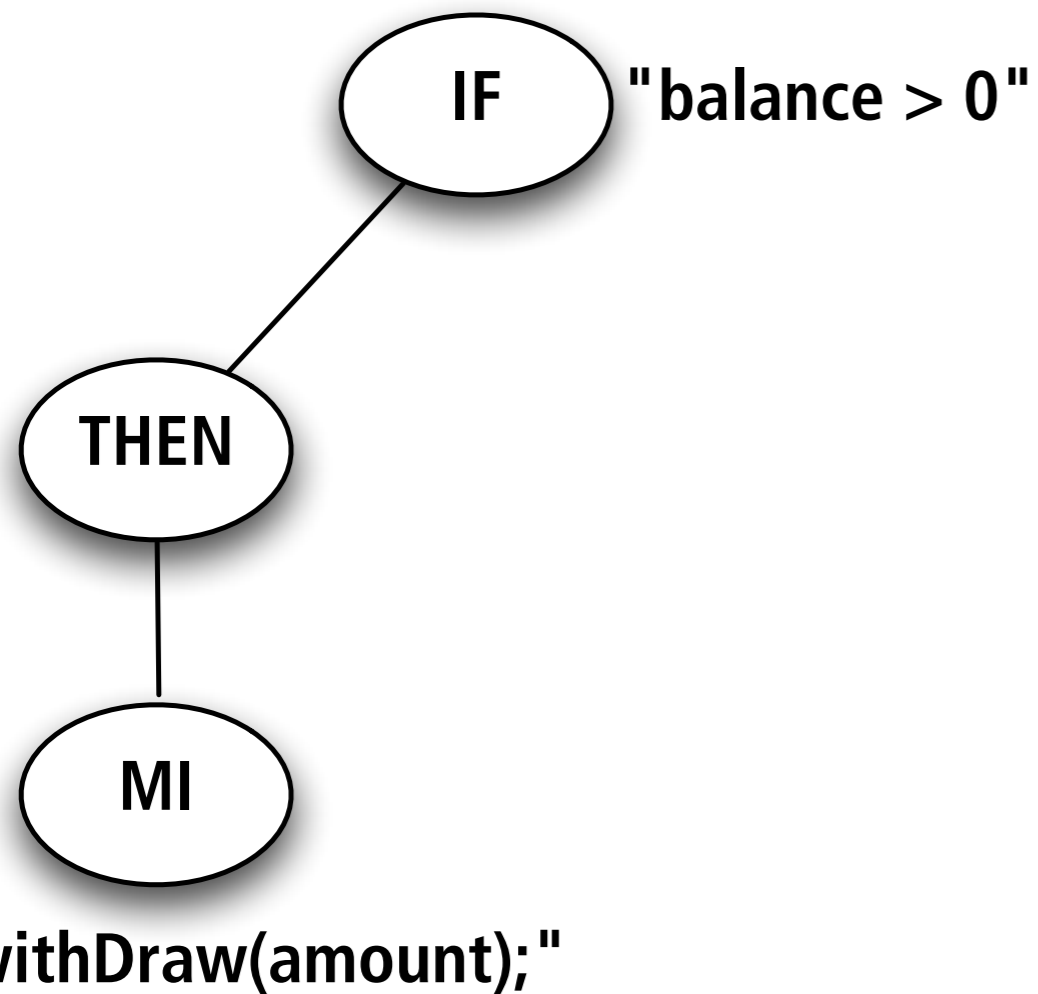
SCC Example

Account.java 1.5

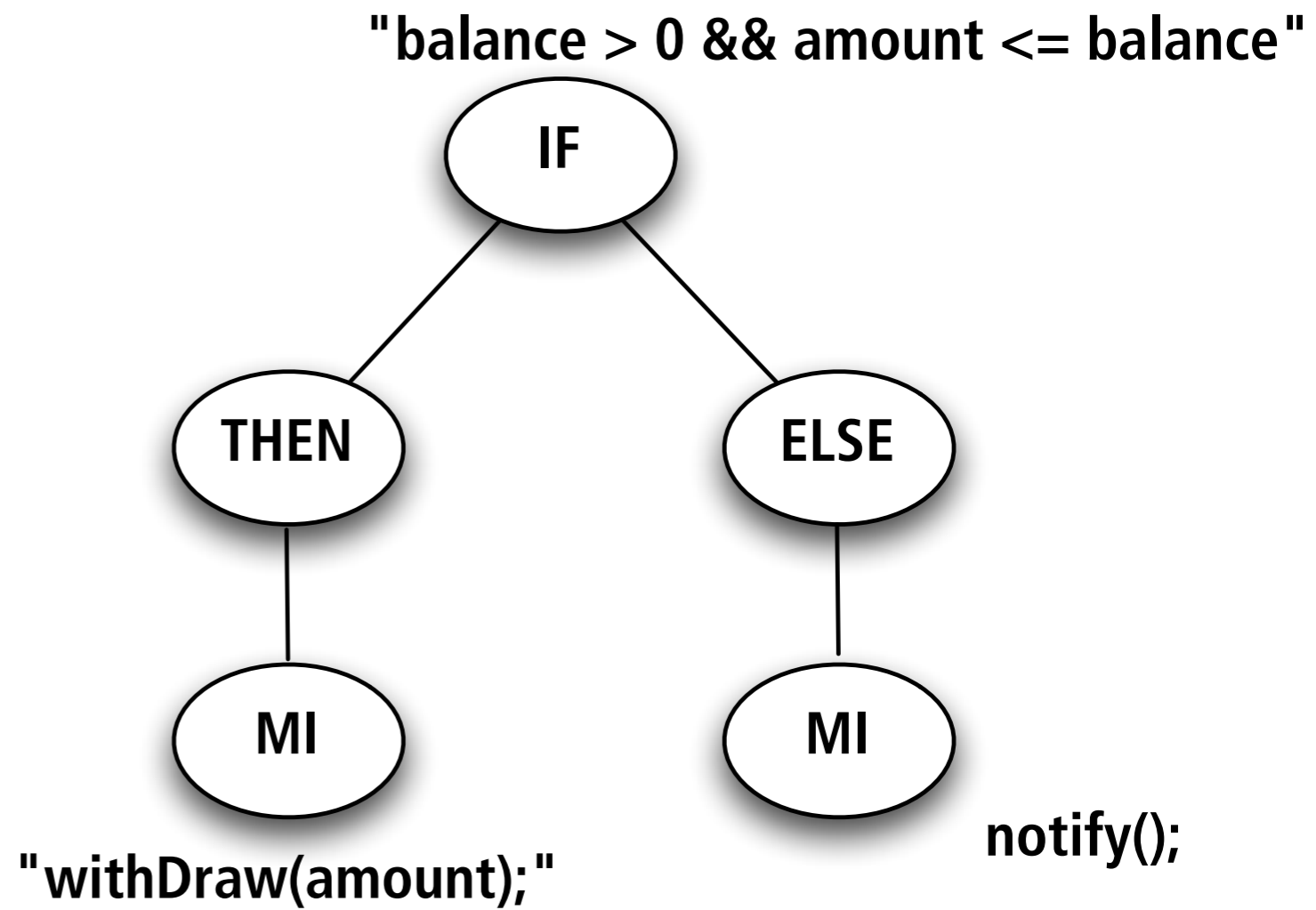


SCC Example

Account.java 1.5

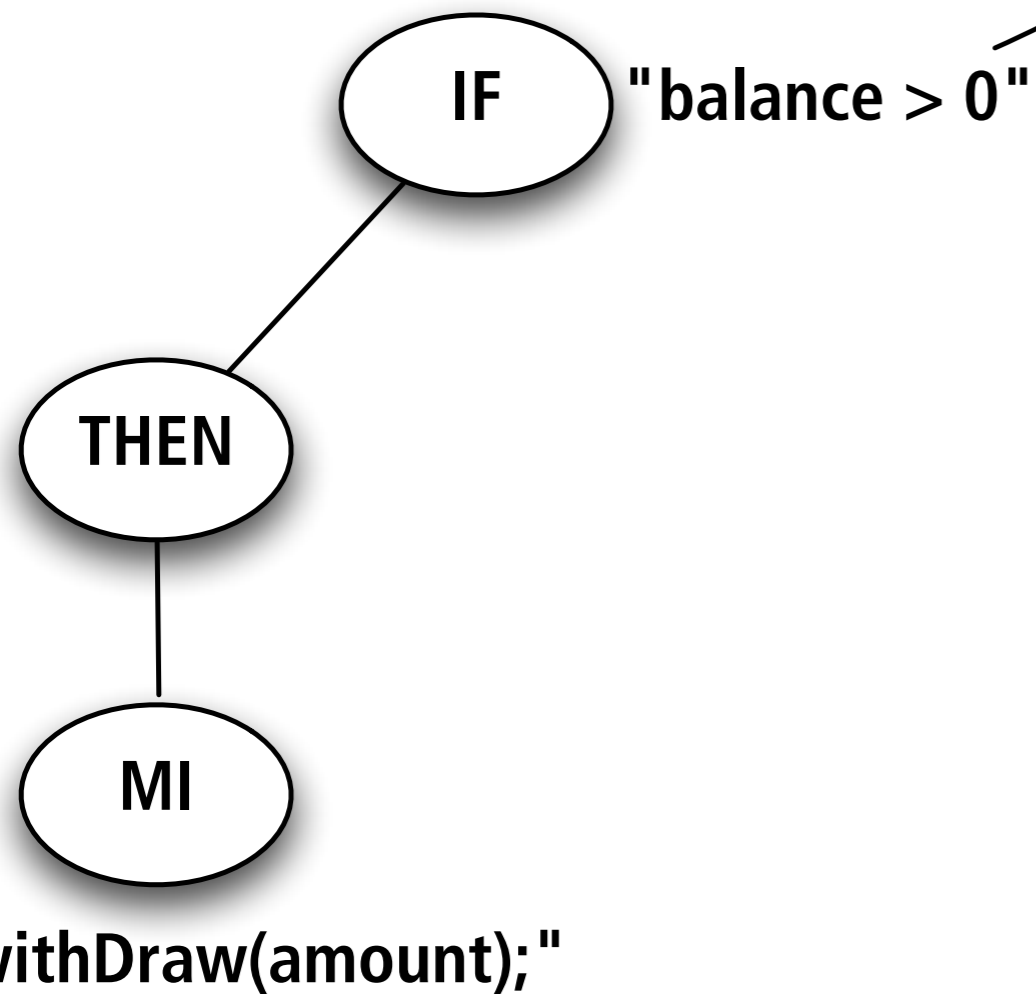


Account.java 1.6

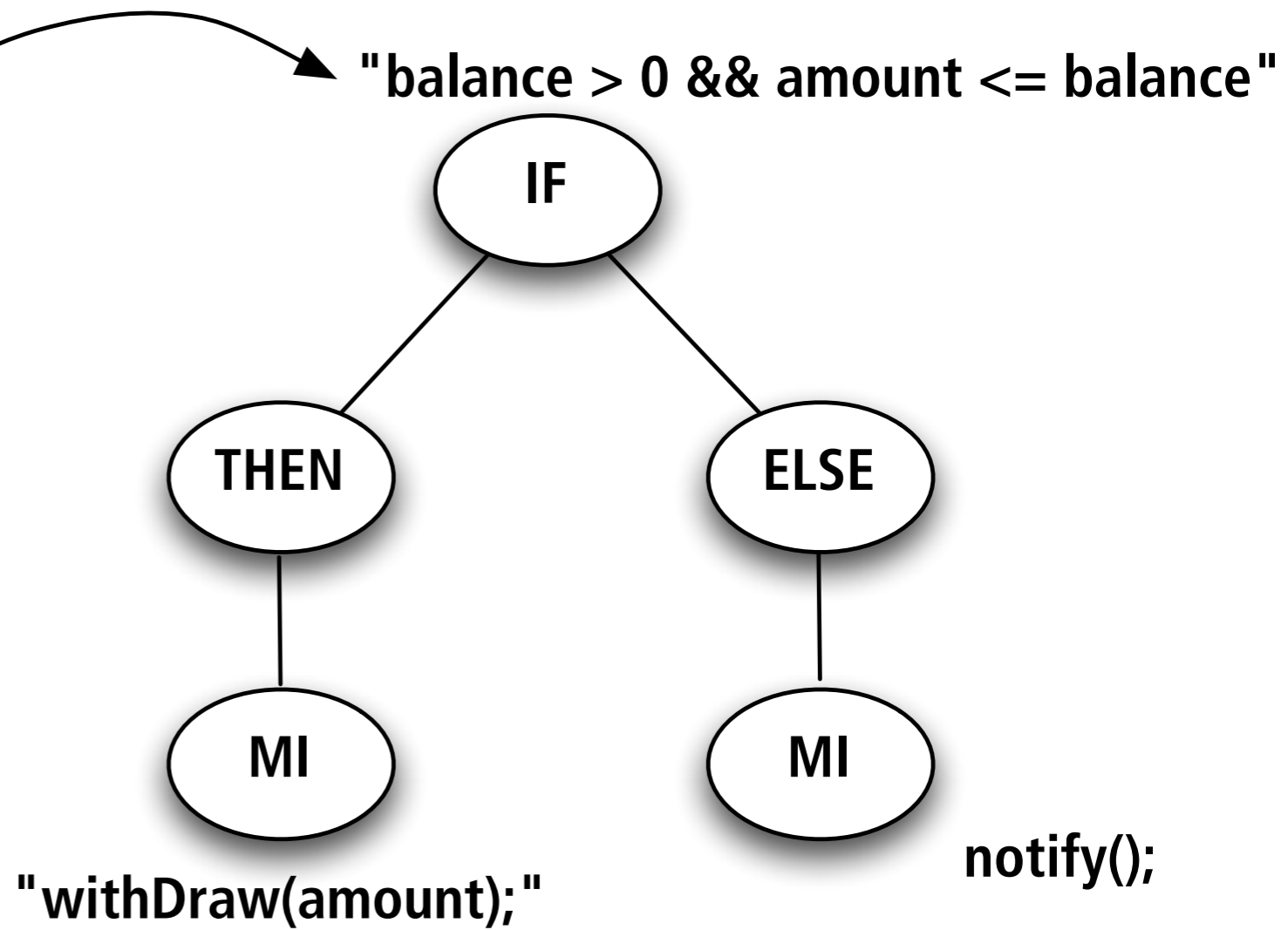


SCC Example

Account.java 1.5

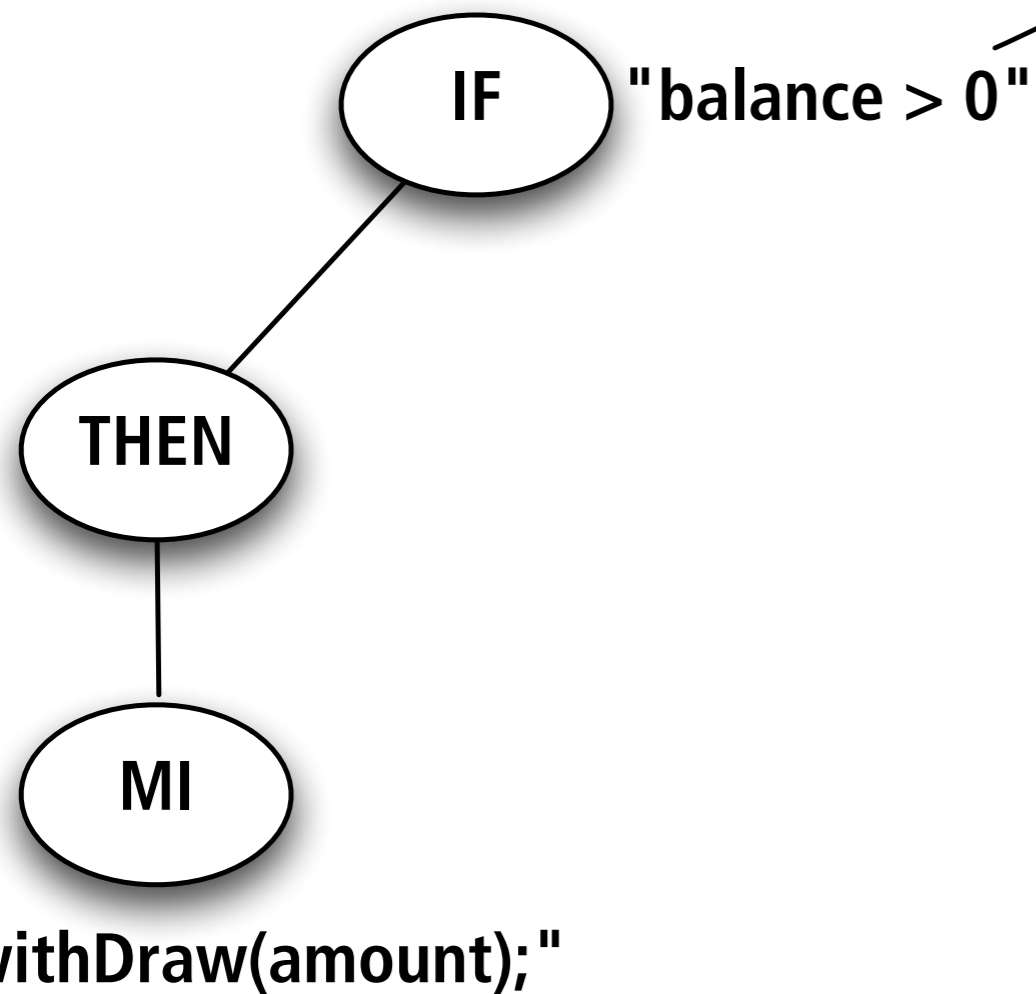


Account.java 1.6

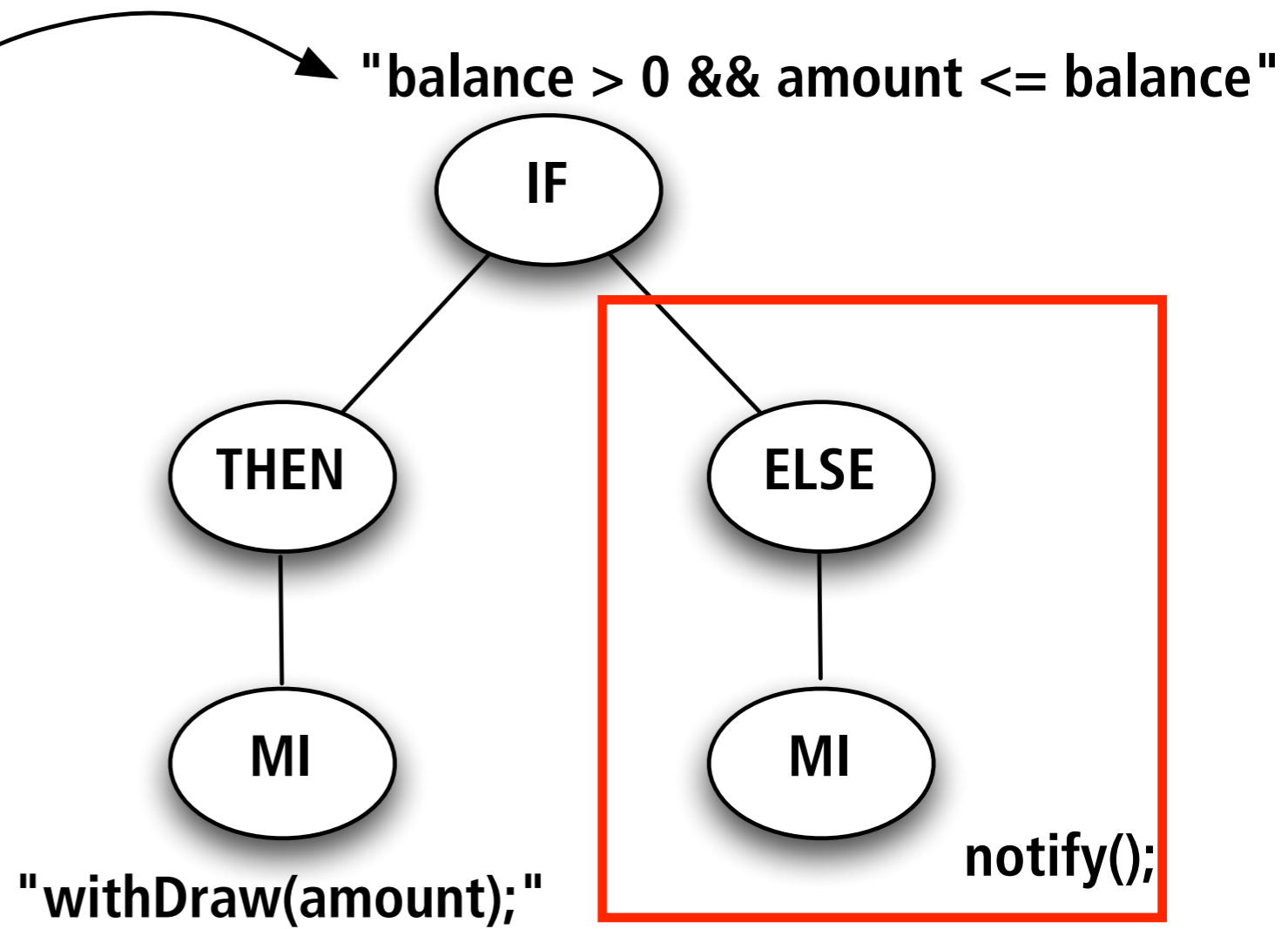


SCC Example

Account.java 1.5

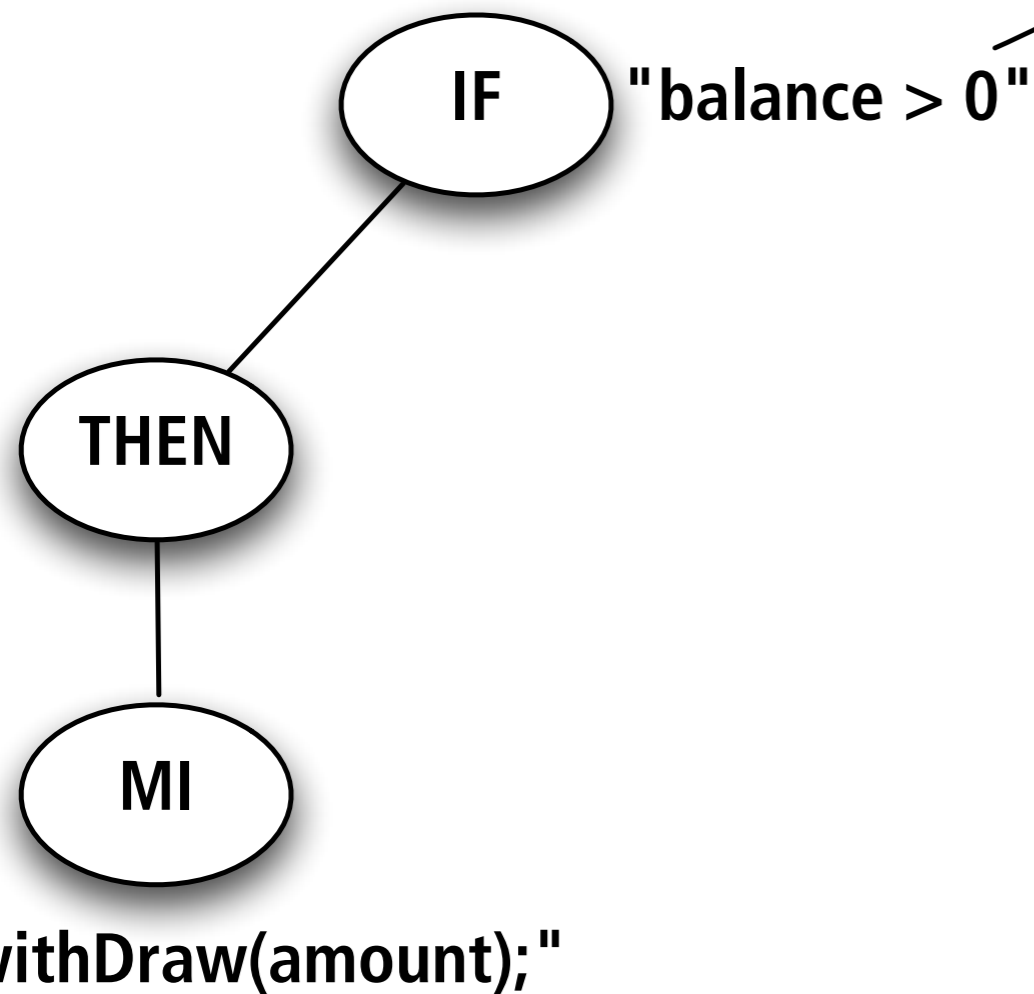


Account.java 1.6

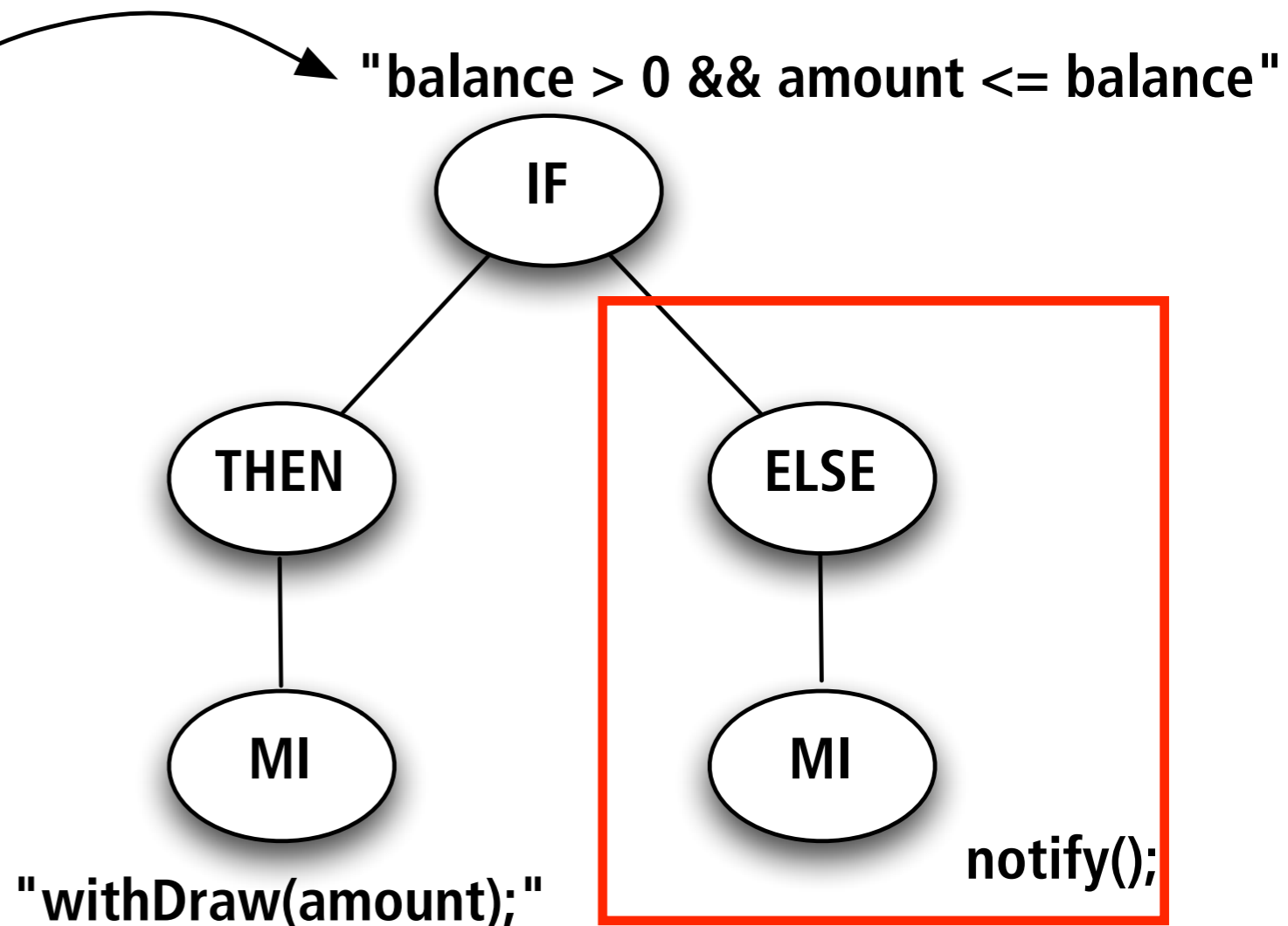


SCC Example

Account.java 1.5



Account.java 1.6



3xSCC: 1x condition change, 1x else-part insert, 1x invocation statement insert

Empirical Studies

- Study 1: Correlation of the number of bugs with SCC and Code Churn on file level
- Study 2: Can SCC be used to identify *bug-prone* files? How do SCC compare with Code Churn?
- Study 3: Can SCC be used to predict the number of bugs in a file? How do SCC compare with Code Churn?

Dataset

- 15 Eclipse Plugins
- ca. 850'000 fine-grained source code changes (SCC)
- ca. 10'000 files
- ca. 9'700'000 lines modified (LM)
- ca. 9 years of development history
- and a lot of bugs
- Bug references in commit messages

Study 2: Bug Prone?

- *Bug-prone vs not bug-prone*
- A priori binning using the median
- Different binning cut points = different prior probabilities
- Area under the curve (AUC)

$$bugClass = \begin{cases} not\ bug - prone & : \ #bugs \leq median \\ bug - prone & : \ #bugs > median \end{cases}$$

Study 2: Bug Prone?

- Prediction Experiment 1:
- Logistic Regression with the number of LM and SCC per file as predictors
- Logistic Regression = non linear regression when dependent variable is dichotomous

Study 2: Bug Prone?

- Results of Prediction Experiment 1:
- LM and SCC are good predictor with average AUC of 0.85 and 0.9
- Related Samples Wilcoxon Signed-Ranks Test on the AUC values of LM and SCC was significant at $\alpha = 0.01$
- SCC has significantly higher AUC values in our dataset

Study 2: Bug Prone?

- Prediction Experiment 2: Using change types as predictors
- There are large differences in the frequencies of change types, i.e. how often certain change types occurs
- We used the following change type categories: cDecl, func, oState, mDecl, stmt, cond, else
- 8 different machine learning algorithms

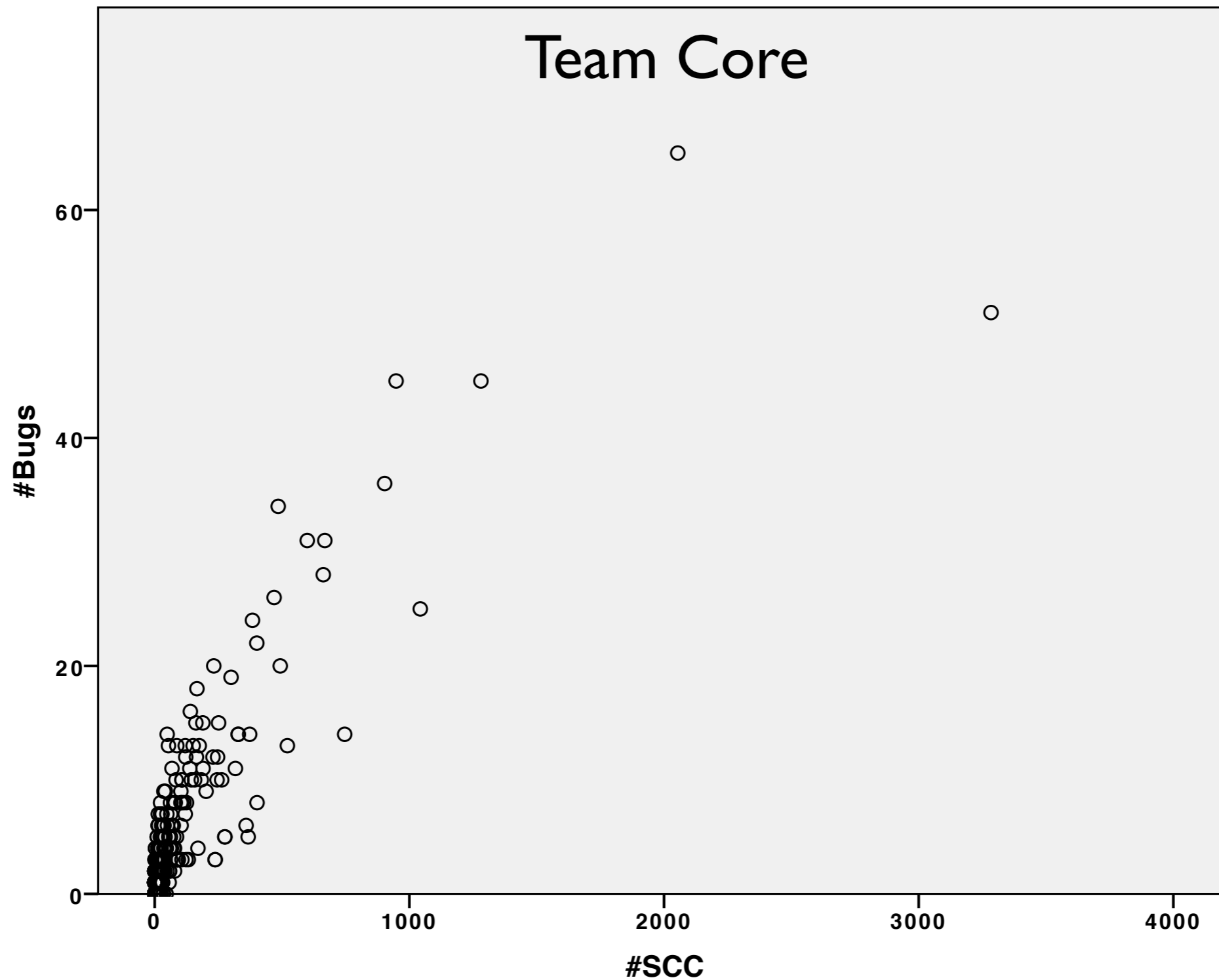
Study 2: Bug Prone?

- Results of Prediction Experiment 2:
- Change type categories are good indicators of *bug-prone* files.
- Some classifiers such as, e.g. SVM (avg. AUC of 0.88), perform explicitly well (as possibly better as well)
- But statistical test show that the better performance is not necessarily significant
- The knowledge of change types of categories does not improve performance

Study 3: Number of Bugs?

- Predicting the number of bugs in files using LM and SCC
- What kind of function fits and describes the relation of the number of bugs with LM and SCC the best?
- Linear, Cubic,

Study 3: Number of Bugs?



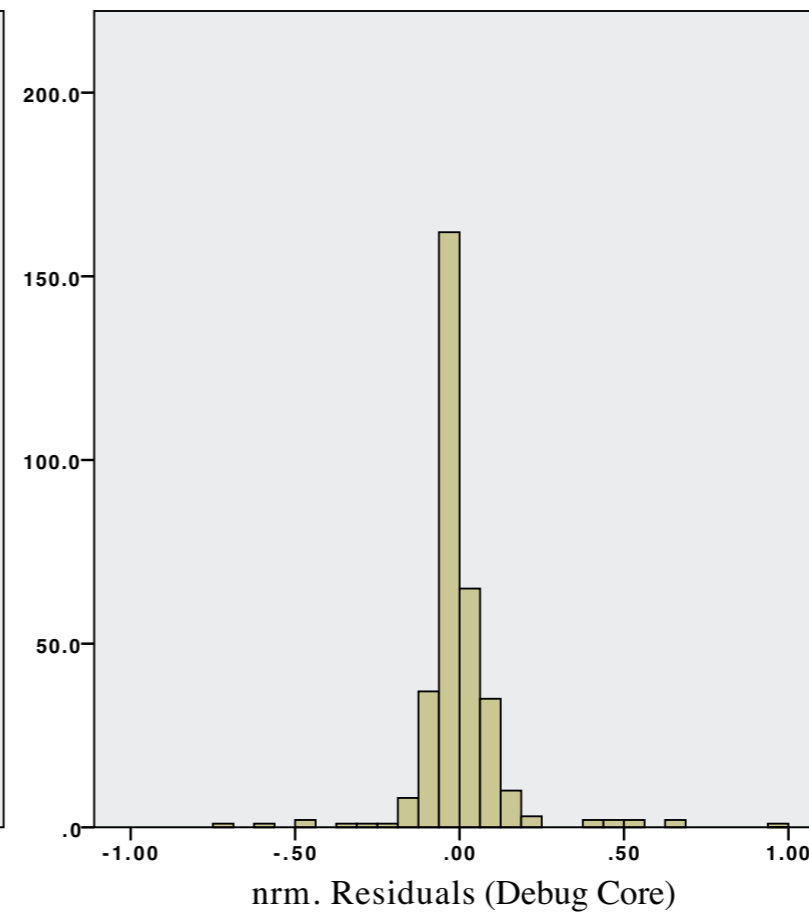
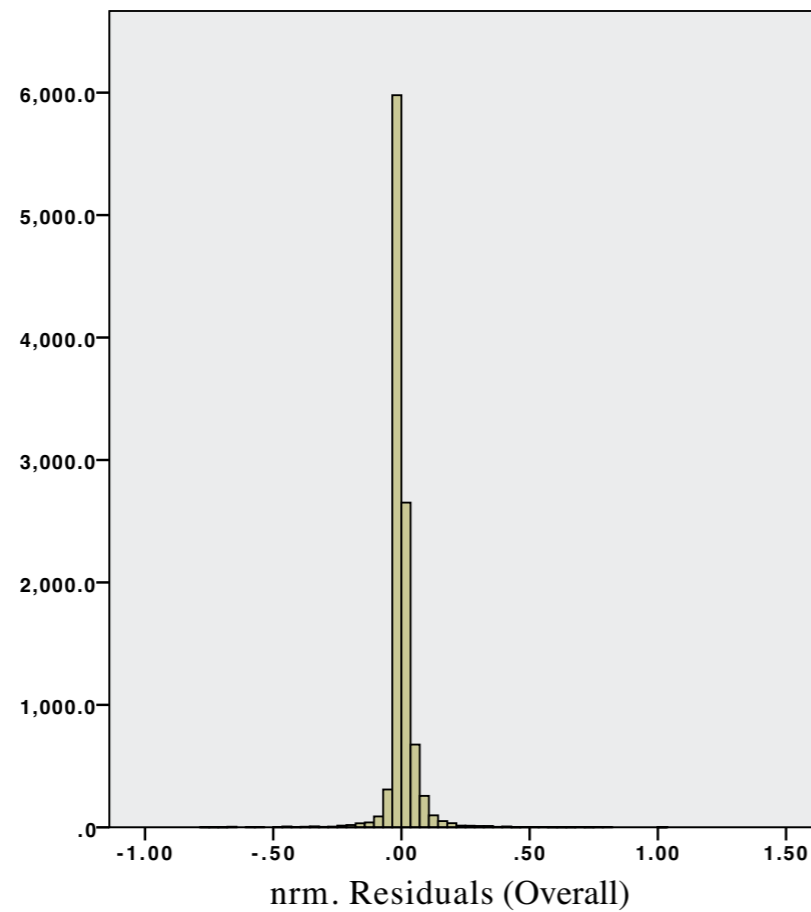
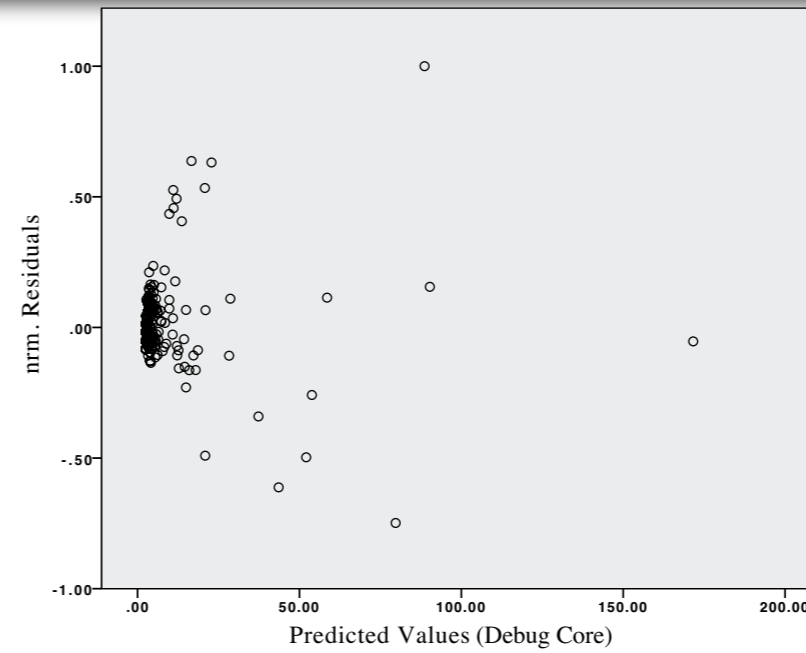
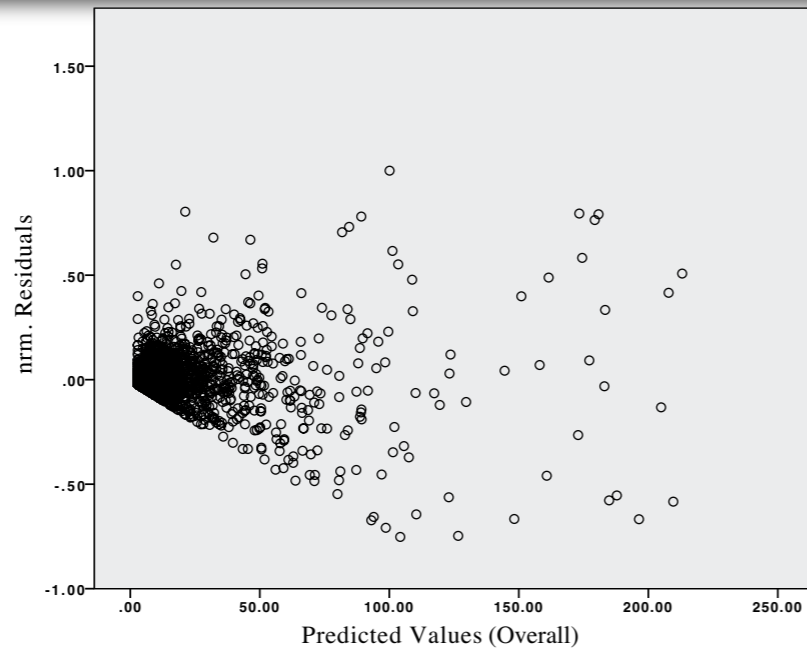
Study 3: Number of Bugs?

- Non linear regression with asymptotic model:
- $f(\text{bugs}) = a_1 + b_2 * e^{b_3 * SCC}$
- Using this function we model a saturation effect
- This is similar to Logistic Regression

Study 3: Number of Bugs?

- Results:
- Adequate explanatory power
- Average R2: LM 0.7 vs. SCC 0.79
- Related Samples Wilcoxon Signed-Ranks Test on the R2 values of LM and SCC was significant at $\alpha = 0.01$
- SCC has a significantly higher R2 values in our dataset
- Error terms?

Error Terms



Conclusions

- SCC is a good predictor
- SCC is significant better than LM
- Advanced learners are better, but not always significant
- Change types do not yield extra discriminatory power
- Predicting the number of bugs is possible to some extent - But: *Be careful!*