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

Working Conference on Mining Software Repositories 2011

Emanuel Giger[1], Martin Pinzger[2], Harald Gall[1]

[1]University of Zurich, Switzerland
[2]Delft University of Technology, The Netherlands

University of Zurich
Department of Informatics

Bug Prediction

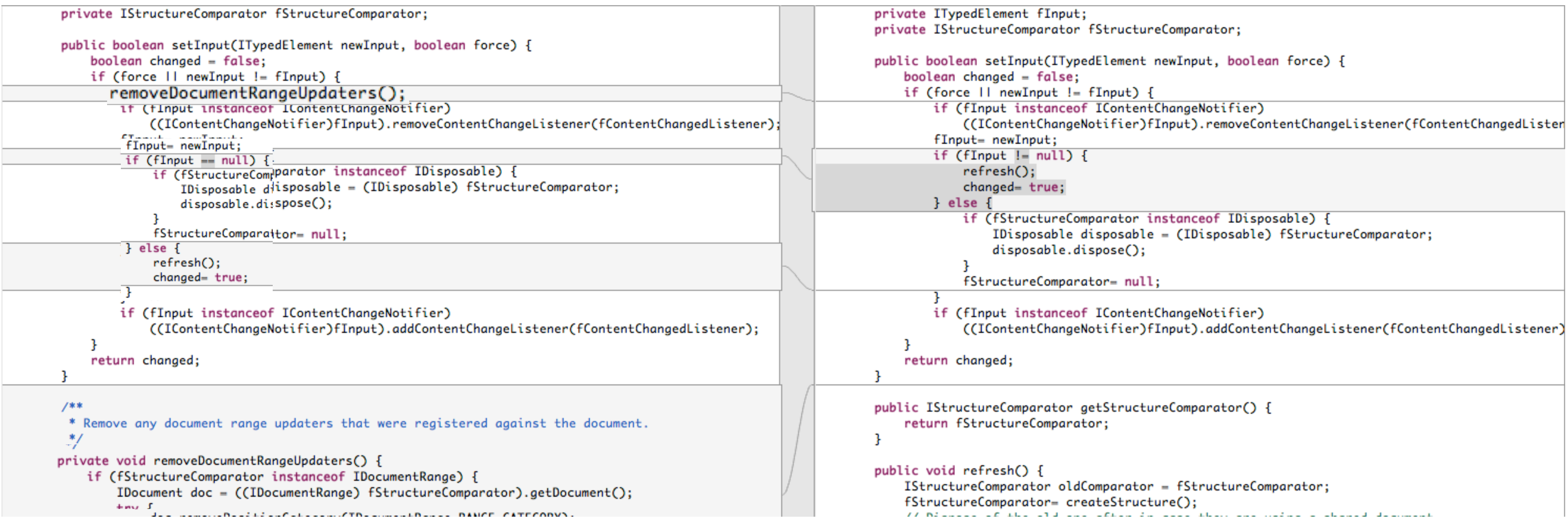
- Many useful papers on building bug prediction models
- Product measures, process measures, organizational measures - or a combination
- Process measures performed particularly well
- Very popular: Revisions and Code Churn

Change Measures

- File Revisions
- Code Churn aka Lines added/deleted/changed
- Both provided by Software Repositories
- Various ways to measure them: relative, consecutive, timeframes.....

Revisions are coarse grained

There is more than just a *file revision*



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.*;


/**
```

Renaming is an example

- local variable: int limit = 65; to int speedLimit = 65;
- public Point getXYCoordinates(){...} to

  public Point get2DCoordinates(){...} and then

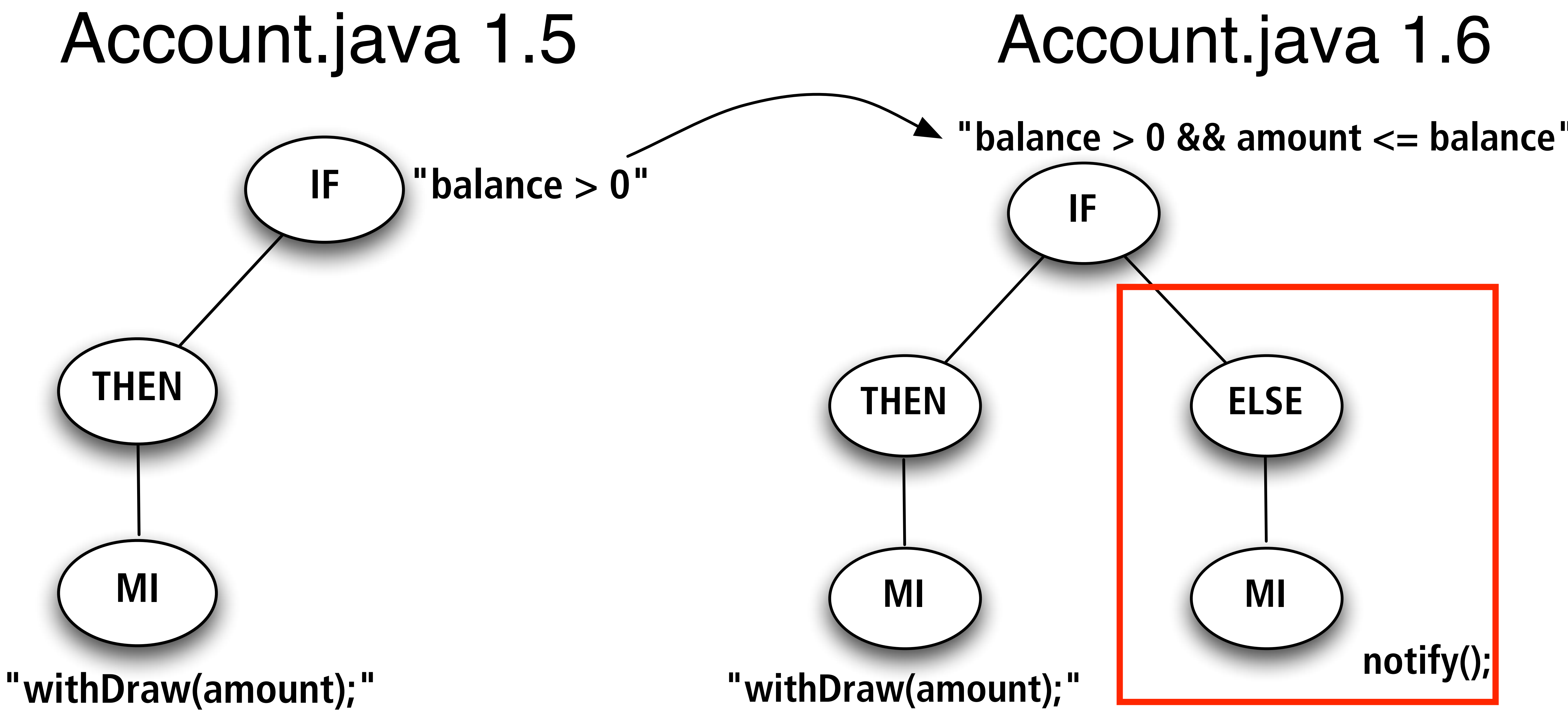
  public 2DPoint get2DCoordinates(2DPoint){...}
- Each time the Versioning System will likely report "**1 line changed**"

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 file[1]

[1]Beat 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



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

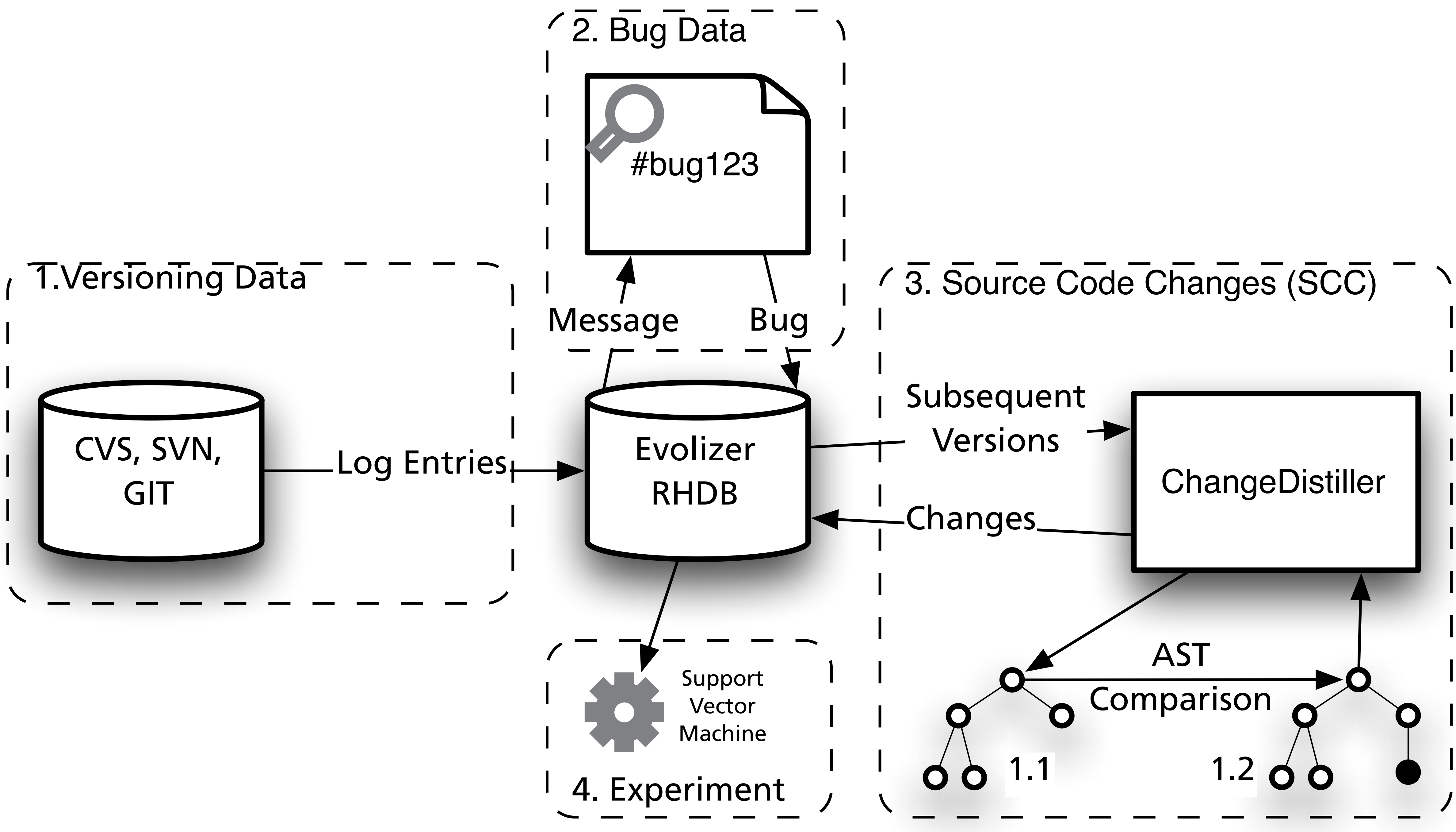
Empirical Studies

- 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

Approach



2. Bug Data
#bug123
Message
Bug
1.Versioning Data
CVS, SVN, GIT
Log Entries
Evolizer RHDB
3. Source Code Changes (SCC)
Subsequent Versions
ChangeDistiller
Changes
AST Comparison
1.1
1.2
Support Vector Machine
4. Experiment

Study 1: Correlation

- +/-0.5 substantial
- +/-0.7 strong

Spearman rank correlation between Bugs and LM, SCC (* = significant correlation at 0.01)

Eclipse Project	*LM*	*SCC*
Compare	0.68*	**0.76** *
jFace	**0.74** *	0.71*
JDT Debug	0.62*	**0.8** *
Resource	0.75*	**0.86** *
Runtime	0.66*	**0.79** *
Team Core	0.15*	**0.66** *
CVS Core	0.60*	**0.79** *
Debug Core	0.63*	**0.78** *
jFace Text	**0.75** *	0.74*
Update Core	0.43*	**0.62** *
Debug UI	0.56*	**0.81** *
JDT Debug UI	0.80*	**0.81** *
Help	**0.54** *	0.48*
JDT Core	0.70*	**0.74** *
OSGI	0.70*	**0.77** *
Median	0.66	0.77

Study 1: Correlation

- What about the type of changes?
- There are large differences in the frequencies of change types, i.e. how often a certain change type occurs
- We used the following change type categories: cDecl, func, oState, mDecl, stmt, cond, else

Study 1: Correlation

Eclipse Project	cDecl	oState	func	mDecl	stmt	cond	else
Compare	0.01	0.06	0.08	0.05	0.74	0.03	0.03
jFace	0.02	0.04	0.08	0.11	0.70	0.02	0.03
JDT Debug	0.02	0.06	0.08	0.10	0.70	0.02	0.02
Resource	0.01	0.04	0.02	0.11	0.77	0.03	0.02
Runtime	0.01	0.05	0.07	0.10	0.73	0.03	0.01
Team Core	0.05	0.04	0.13	0.17	0.57	0.02	0.02
CVS Core	0.01	0.04	0.10	0.07	0.73	0.02	0.03
Debug Core	0.04	0.07	0.02	0.13	0.69	0.02	0.03
jFace Text	0.04	0.03	0.06	0.11	0.70	0.03	0.03
Update Core	0.02	0.04	0.07	0.09	0.74	0.02	0.02
Debug UI	0.02	0.06	0.09	0.07	0.70	0.03	0.03
JDT Debug UI	0.01	0.07	0.07	0.05	0.75	0.02	0.03
Help	0.02	0.05	0.08	0.07	0.73	0.02	0.03
JDT Core	0.00	0.03	0.03	0.05	0.80	0.05	0.04
OSGI	0.03	0.04	0.06	0.11	0.71	0.03	0.02
Mean	0.02	0.05	0.07	0.09	0.72	0.03	0.03
Variance	0.000	0.000	0.001	0.001	0.003	0.000	0.000

Relative frequencies of SCC categories per Eclipse project, plus their mean and variance over all selected projects.

Study 1: Correlation

Eclipse Project	cDecl	oState	func	mDecl	stmt	cond	else
Compare	0.54*	0.61*	0.67*	0.61*	0.66*	0.55*	0.52*
jFace	0.41*	0.47*	0.57*	0.63*	0.66*	0.51*	0.48*
Resource	0.49*	0.62*	0.7*	0.73*	0.67*	0.49*	0.46*
Team Core	0.44*	0.43*	0.56*	0.52*	0.53*	0.36*	0.35*
CVS Core	0.39*	0.62*	0.66*	0.57*	0.72*	0.58*	0.56*
Debug Core	0.45*	0.55*	0.61*	0.51*	0.59*	0.45*	0.46*
Runtime	0.47*	0.58*	0.66*	0.61*	0.66*	0.55*	0.45*
JDT Debug	0.42*	0.45*	0.56*	0.55*	0.64*	0.46*	0.44*
jFace Text	0.50*	0.55*	0.54*	0.64*	0.62*	0.59*	0.55*
JDT Debug UI	0.46*	0.57*	0.62*	0.53*	0.74*	0.57*	0.54*
Update Core	0.63*	0.4*	0.43*	0.51*	0.45*	0.38*	0.39*
Debug UI	0.44*	0.50*	0.63*	0.60*	0.72*	0.54*	0.52*
Help	0.37*	0.43*	0.42*	0.43*	0.44*	0.36*	0.41*
JDT Core	0.39*	0.6*	0.69*	0.70*	0.67*	0.62*	0.6*
OSGI	0.47*	0.6*	0.66*	0.65*	0.63*	0.57*	0.48*
Mean	0.46	0.53	0.6	0.59	0.63	0.51	0.48
Median	0.45	0.55	0.62	0.60	0.66	0.54	0.48

Spearman rank correlation between Bugs and SCC categories per Eclipse project (* = correlation at 0.01)

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)

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

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?

Eclipse Project	AUC LM	AUC SCC
Compare	0.84	**0.85**
jFace	0.90	0.90
JDT Debug	0.83	**0.95**
Resource	0.87	**0.93**
Runtime	0.83	**0.91**
Team Core	0.62	**0.87**
CVS Core	0.80	**0.90**
Debug Core	0.86	**0.94**
jFace Text	0.87	0.87
Update Core	0.78	**0.85**
Debug UI	0.85	**0.93**
JDT Debug UI	0.90	**0.91**
Help	**0.75**	0.70
JDT Core	0.86	**0.87**
OSGI	0.88	0.88
Median	0.85	**0.90**
Overall	0.85	**0.89**

AUC using logistic regression with LM and SCC to classify source files into bug- prone or not bug-prone.

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

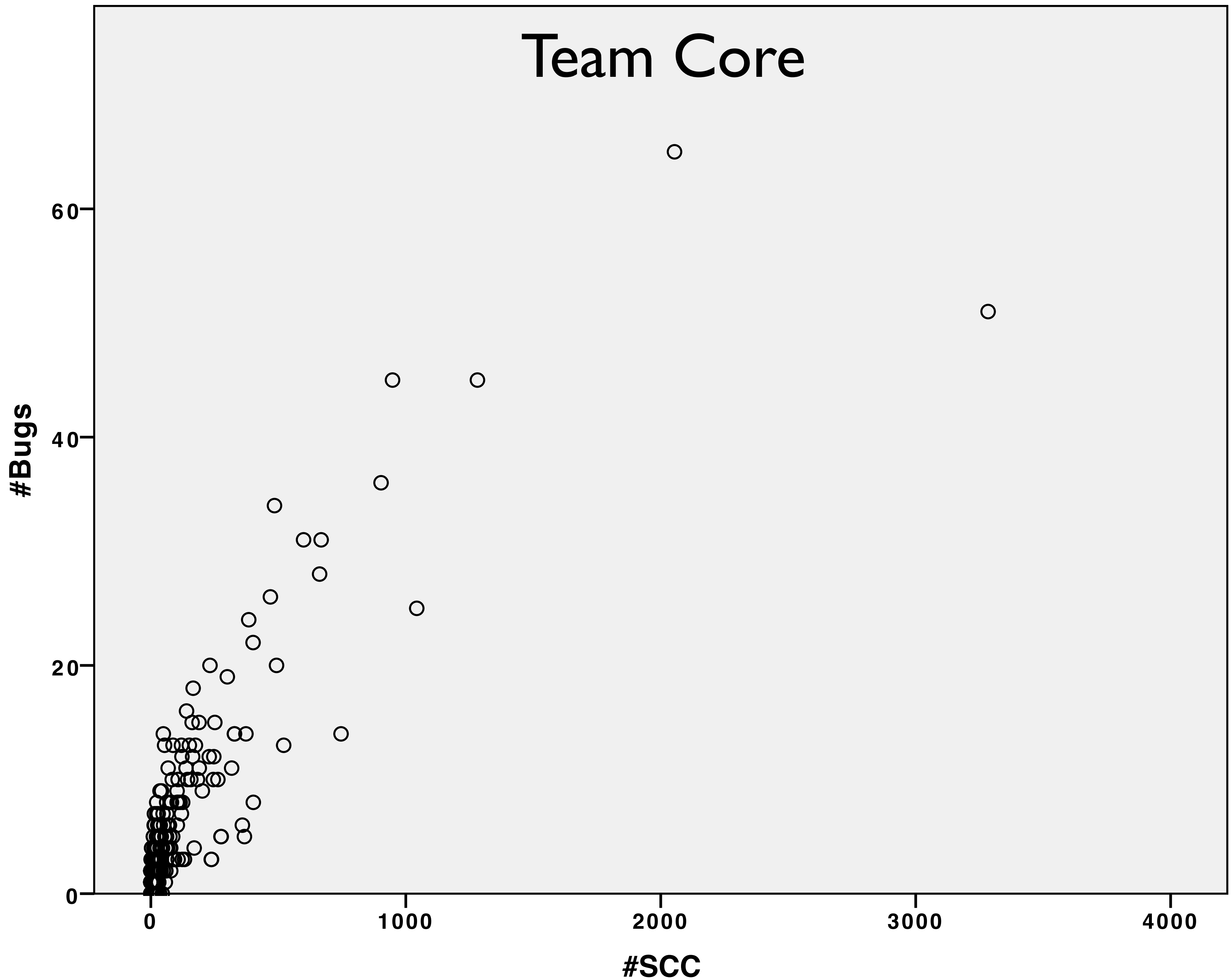
Study 2: Bug Prone?

- 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:
- $f(bugs) = a1 + b2*e^{b3*SCC}$
- Using this function we model a saturation effect
- This is similar to Logistic Regression

Study 3: Number of Bugs?

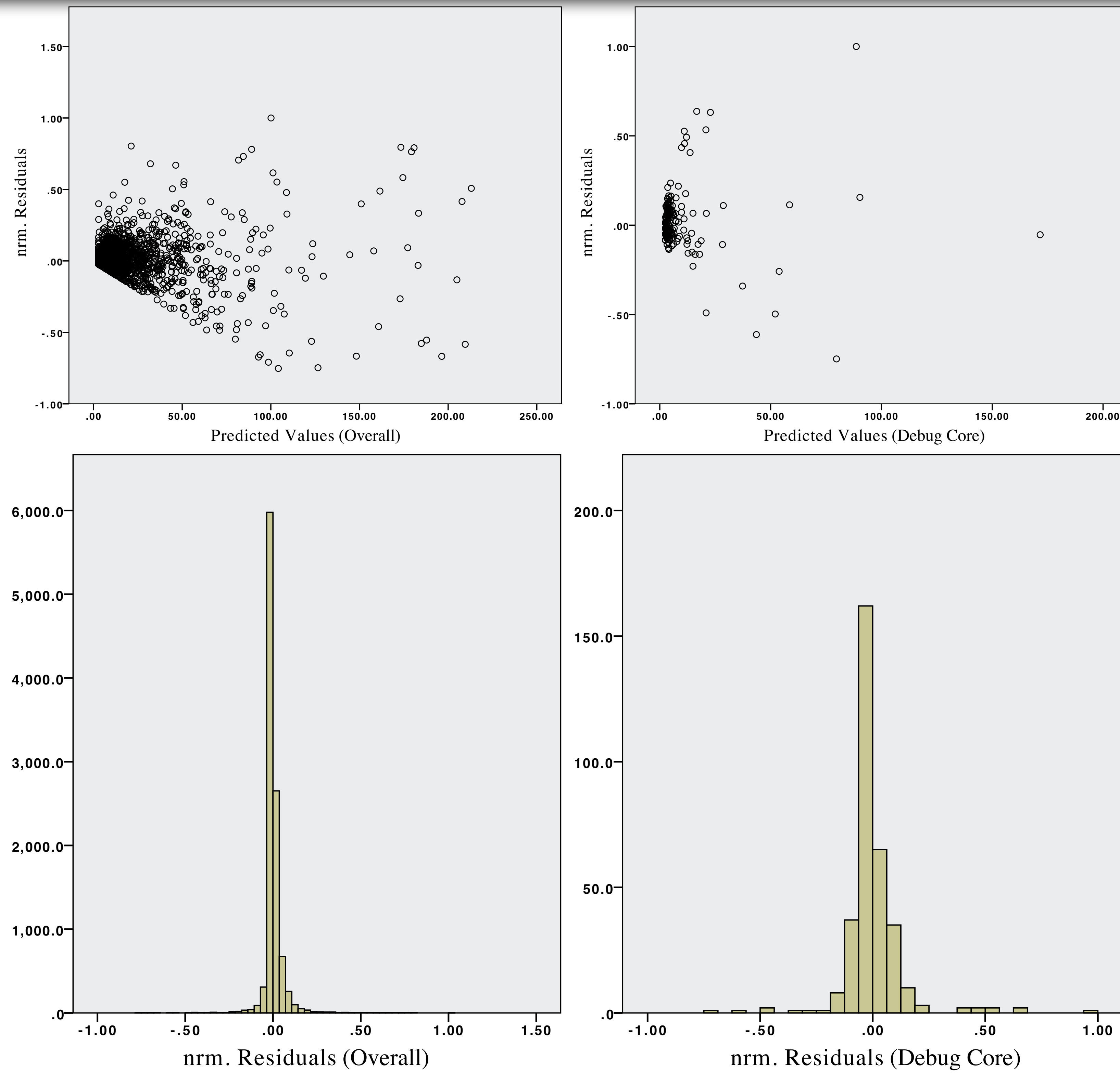
Project	R^2_{LM}	R^2_{SCC}	$Spearman_{LM}$	$Spearman_{SCC}$
Compare	0.84	**0.88**	0.68	**0.76**
jFace	0.74	**0.79**	**0.74**	0.71
JDT Debug	0.69	**0.68**	0.62	**0.8**
Resource	0.81	**0.85**	0.75	**0.86**
Runtime	0.69	**0.72**	0.66	**0.79**
Team Core	0.26	**0.53**	0.15	**0.66**
CVS Core	0.76	**0.83**	0.62	**0.79**
Debug Core	0.88	**0.92**	0.63	**0.78**
Jface Text	0.83	**0.89**	**0.75**	0.74
Update Core	0.41	**0.48**	0.43	**0.62**
Debug UI	0.7	**0.79**	0.56	**0.81**
JDT Debug UI	0.82	0.82	0.8	**0.81**
Help	0.66	**0.67**	0.54	**0.84**
JDT Core	0.69	**0.77**	0.7	**0.74**
OSGI	0.51	**0.8**	0.74	**0.77**
Median	0.7	**0.79**	0.66	**0.77**
Overall	0.65	**0.72**	0.62	**0.74**

Results of the nonlinear regression in terms of R2 and Spearman correlation using LM and SCC as predictors.

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



nrm. Residuals
Predicted Values (Overall)
nrm. Residuals
Predicted Values (Debug Core)
nrm. Residuals (Overall)
nrm. Residuals (Debug Core)

Study 3: Number of Bugs?

- Asymptotic Model: Adequate Results
- Check regression assumptions
- Probably *as good as it gets* given the data
- Segmented Regression?

Conclusions

- 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 extend - But: Be *careful!*

Paper: Comparing Fine-Grained Source Code Changes And Code Churn For Bug Prediction, E. Giger, M. Pinzger, and H. C. Gall, MSR 2011, pp. 83-92, ACM, IEEE CS Press, 2011.