eROSE: Guiding Programmers in Eclipse

Thomas Zimmermann · Valentin Dallmeier · Konstantin Halachev · Andreas Zeller
Department of Computer Science
Saarland University
Saarbrücken, Germany
{zimmerth, dallmeier, halachev, zeller}@st.cs.uni-sb.de

ABSTRACT

The eROSE plugin for ECLIPSE uses version archives to guide programmers: You changed fKeys[] and eROSE suggests you to change initDefaults(), because both items always have been changed together in the past. In addition, eROSE points out item coupling that is undetectable by program analysis, for instance between fKeys[] and a properties file. All eROSE needs is a CVS repository; we designed eROSE to be as efficient and unobtrusive as possible.

Categories and Subject Descriptors

D.2.6 [**Programming Environments**]: Integrated environments; D.2.7 [**Distribution, Maintenance, and Enhancement**]: Version control; H.2.8 [**Database Applications**]: Data mining

General Terms

Management

Keywords

Software evolution, programming environments, version control, program comprehension, data mining, association rules.

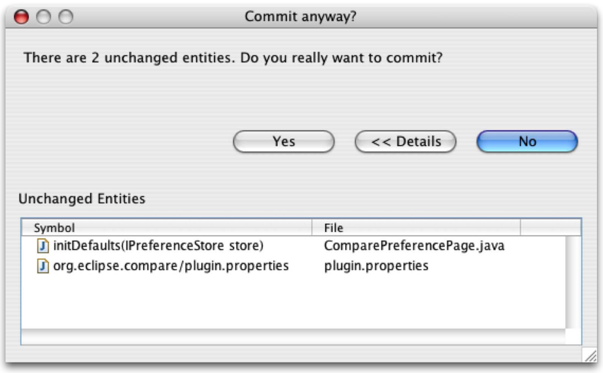
1. INTRODUCTION

Suppose you apply a change to a large software system. What else do you have to change? Missing a required update results in bugs that could easily be avoided.

Just like Amazon.com suggests related products after a purchase, our eROSE plugin for Eclipse guides programmers based on the system history. Suppose you changed an array fKeys[]. eROSE then suggests to change the initDefaults() function—because in the past, both items always have been changed together. If the programmer misses to commit a related change, eROSE issues a warning.

All eROSE needs is a CVS repository; we designed eROSE to be as efficient and unobtrusive as possible. In order to create precise recommendations, eROSE applies efficient data mining techniques: In the Eclipse project with more than 2.9 million lines of code and 300,000 changes, eROSE suggests related changes in less than a second [3].

The benefit of eROSE is that it points out item coupling that is undetectable by program analysis—such as between the fKeys[] array and a properties file, or between an SQL query and an image file that contains the database schema.



Commit anyway?

There are 2 unchanged entities. Do you really want to commit?

Unchanged Entities

Symbol	File
initDefaults(IPreferenceStore store)	ComparePreferencePage.java
org.eclipse.compare/plugin.properties	plugin.properties

Figure 2: eROSE points programmers to locations they have likely missed.

2. EROSE IN A NUTSHELL

eROSE learns from the information stored in version archives to make recommendations for programmers. These recommendations are useful in two common scenarios:

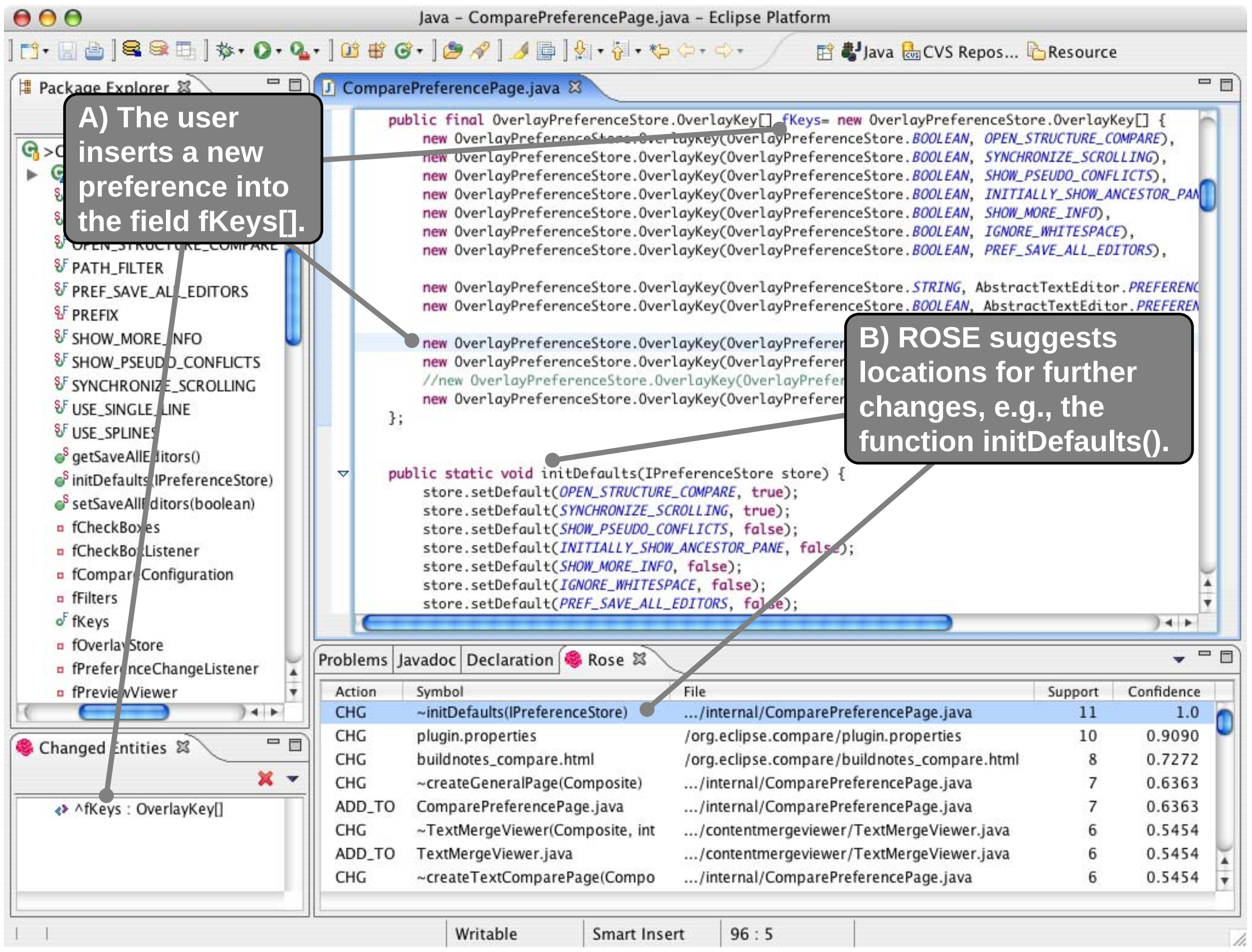
The "Improve Navigation" scenario. The major application for eROSE is to guide users through source code: The user changes some entity and eROSE automatically recommends related changes in a view.

Figure 1 shows our eROSE tool as a plug-in for the ECLIPSE programming environment. The programmer is inserting a new preference, and has added an element to the fKeys[] array. eROSE now suggests to consider further changes, as inferred from the version history. First come the locations with the highest *confidence*—that is, the likelihood that further changes be applied to the given location.

Position 3 on the list is an HTML documentation file with a confidence of 0.727—suggesting that after adding the new preference, the documentation should be updated, too. Such a dependency is undetectable by program analysis.

The "Prevent Errors" scenario. Besides supporting navigation, eROSE should also *prevent errors*. The scenario is that when a user decides to commit changes to the version archive, eROSE checks if there are related changes with a *high confidence* that have not been changed yet.

If there are, like in Figure 1 the top two locations, eROSE issues a pop-up window with a warning. It also suggests the "missing" item that should be considered, as in Figure 2.



```
public final OverlayPreferenceStore.OverlayKey[] fKeys= new OverlayPreferenceStore.OverlayKey[] {
    new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.BOOLEAN, OPEN_STRUCTURE_COMPARE),
    new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.BOOLEAN, SYNCHRONIZE_SCROLLING),
    new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.BOOLEAN, SHOW_PSEUDO_CONFLICTS),
    new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.BOOLEAN, INITIALLY_SHOW_ANCESTOR_PAN
    new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.BOOLEAN, SHOW_MORE_INFO),
    new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.BOOLEAN, IGNORE_WHITESPACE),
    new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.BOOLEAN, PREF_SAVE_ALL_EDITORS),

    new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.STRING, AbstractTextEditor.PREFERENC
    new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.BOOLEAN, AbstractTextEditor.PREFEREN

    new OverlayPreferenceStore.OverlayKey(OverlayPreferer
    new OverlayPreferenceStore.OverlayKey(OverlayPreferer
    //new OverlayPreferenceStore.OverlayKey(OverlayPrefer
    new OverlayPreferenceStore.OverlayKey(OverlayPreferer
};


public static void initDefaults(IPreferenceStore store) {
    store.setDefault(OPEN_STRUCTURE_COMPARE, true);
    store.setDefault(SYNCHRONIZE_SCROLLING, true);
    store.setDefault(SHOW_PSEUDO_CONFLICTS, false);
    store.setDefault(INITIALLY_SHOW_ANCESTOR_PANE, false);
    store.setDefault(SHOW_MORE_INFO, false);
    store.setDefault(IGNORE_WHITESPACE, false);
    store.setDefault(PREF_SAVE_ALL_EDITORS, false);
```

Action	Symbol	File	Support	Confidence
CHG	~initDefaults(IPreferenceStore)	.../internal/ComparePreferencePage.java	11	1.0
CHG	plugin.properties	/org.eclipse.compare/plugin.properties	10	0.9090
CHG	buildnotes_compare.html	/org.eclipse.compare/buildnotes_compare.html	8	0.7272
CHG	~createGeneralPage(Composite)	.../internal/ComparePreferencePage.java	7	0.6363
ADD_TO	ComparePreferencePage.java	.../internal/ComparePreferencePage.java	7	0.6363
CHG	~TextMergeViewer(Composite, int	.../contentmergeviewer/TextMergeViewer.java	6	0.5454
ADD_TO	TextMergeViewer.java	.../contentmergeviewer/TextMergeViewer.java	6	0.5454
CHG	~createTextComparePage(Compo	.../internal/ComparePreferencePage.java	6	0.5454

Figure 1: After the programmer has made some changes to the source (above), eROSE suggests locations (below) where, in similar transactions in the past, further changes were made.

In terms of architecture, eROSE consists of two parts:

Preprocessing takes a complete version archive as input. The archive is mirrored in a database (*data collection*), changes are mapped to entities and transactions (*data preparation*), and finally noise, caused by large transactions, is removed (*data cleaning*). Preprocessing ensures a fast access to all necessary information.

Mining creates association rules [1] from the preprocessed data. Such rules describe implications between software entities, e.g., "If fKeys[] is changed, then initDefaults() is changed, too". It is possible to mine for all rules, but typically eROSE mines only for rules with a particular left-hand side. Thus, mining is speeded up and rules are always up-to-date.

The preprocessing of eROSE is discussed in [2] and the mining is explained and evaluated in detail in [3].

3. DOWNLOAD

eROSE is open source and available for download from:

```
http://www.st.cs.uni-sb.de/softevo/
```

4. ACKNOWLEDGMENTS

This project is funded by the Deutsche Forschungsgemeinschaft, grant Ze 509/1-1. The development of eROSE as an ECLIPSE plug-in was made possible by an IBM ECLIPSE innovation grant. Special thanks to Stephan Diehl and Peter Weißgerber for the great collaboration and the fruitful discussions. Thanks to Ken Bauer for organizing the OOPSLA 2005 demonstrations.

5. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th Very Large Data Bases Conference (VLDB)*, pages 487–499. Morgan Kaufmann, 1994.

[2] T. Zimmermann and P. Weißgerber. Preprocessing CVS data for fine-grained analysis. In *Proceedings of International Workshop on Mining Software Repositories (MSR 2004)*, pages 2–6, Edinburgh, Scotland, UK, May 2004.

[3] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6):429–445, June 2005.