2 Data Mining Version Histories

Thomas Zimmermann
Andreas Zeller
Lehrstuhl für Softwaretechnik, Universität des Saarlandes, Saarbrücken, Germany
`{tz,zeller}@acm.org`

Abstract

Program analysis long has been understood as the analysis of source code alone. A modern software product, though, is more than just program code; it contains documentation, interface descriptions, resource data—all of which must be maintained and organized. In this paper, we propose a novel approach to maintain such non-program entities: By learning from the development history of the product, we can determine coupling between entities: "Programmers who changed *ComparePreferencePage.java* typically also changed *plugin.properties*". As a first proof of concept, our ROSE plug-in for ECLIPSE automatically guides the programmer along related changes.

2.1 Learning from History

Shopping for a book at Amazon.com, you may have come across a section that reads "Customers who bought this book also bought...", listing other books that were typically included in the same purchase. Such information is gathered by *data mining*— the automated extraction of hidden predictive information from large data sets. We have applied such data mining to the *version histories* of large open-source software systems. This results in rules like the following:

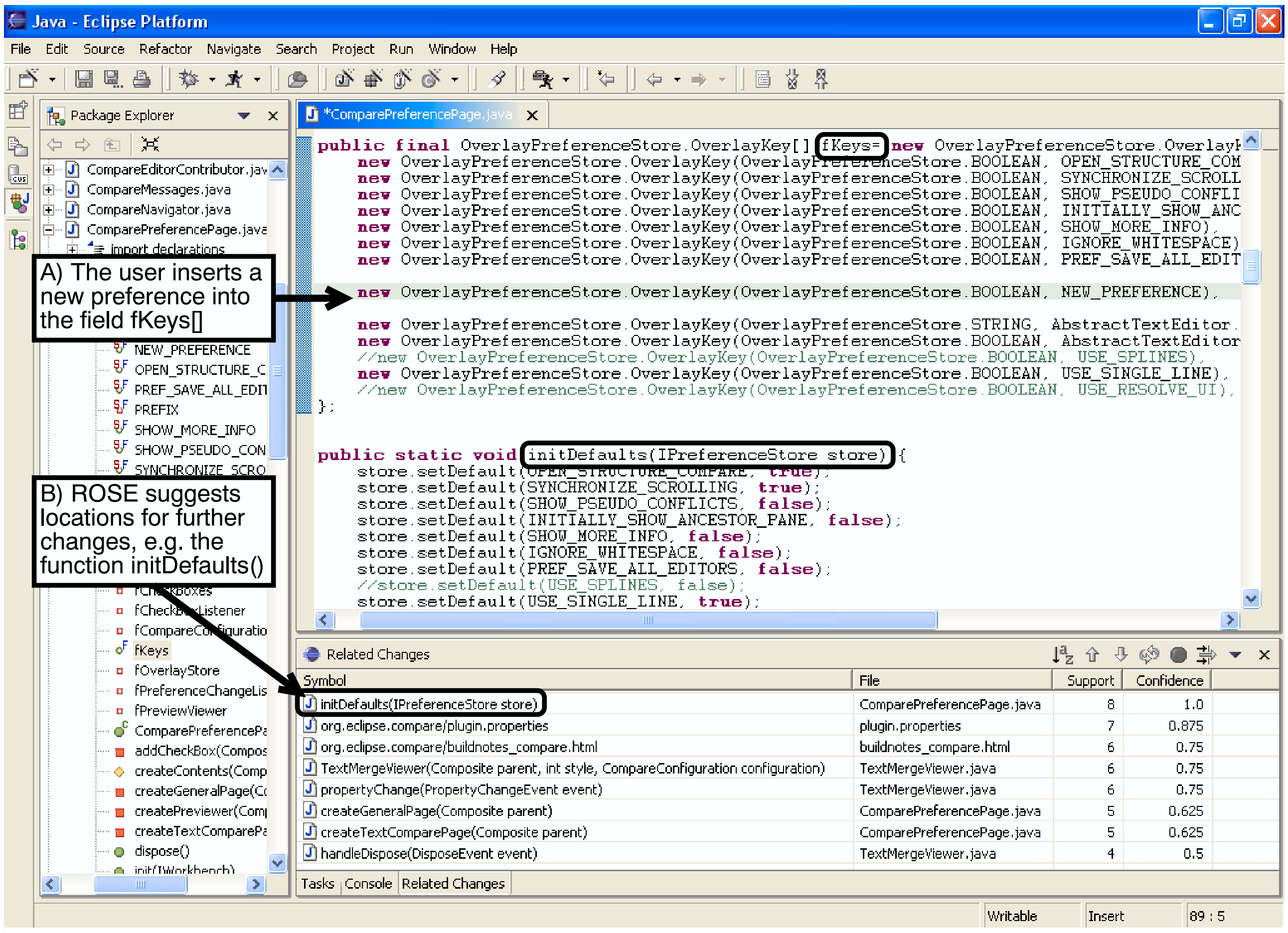
Coupling between entities: *"Programmers who changed the* `fkeys[]` *field always also changed the* `initDefaults()` *function".* The `initDefaults()` function initializes new elements of the `fkeys[]` field; whenever `fkeys[]` was extended by a new element, `initDefaults()` was extended by a statement that initialized the element.

Coupling between programs and documentation: *"In 8 out of 10 cases, Programmers who changed the embedded SQL statement in line 47 of status.py changed the JPEG image igordb.jpg".* The JPEG image is part of the product documentation and is a view of the database schema; whenever the schema changed, the SQL statements were changed, too, and the documentation was updated.

Such rules can reveal invariants of the development process (such as updating documentation); they can reveal factual coupling through common changes; and they can be put to use for actual programmers. Figure 2.1 shows our ROSE plug-in for the ECLIPSE programming environment, actually working on the ECLIPSE source code: As soon as the programmer makes a change to `fkeys[]`, ROSE suggests further related changes as listed above.

2.2 Mining Rules

Figure 2.2 shows the basic information flow through ROSE. The *ROSE Server* first extracts the *transactions* from the CVS archive—changes that were committed by the same



```
public final OverlayPreferenceStore.OverlayKey[] fKeys= new OverlayPreferenceStore.OverlayK
    new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.BOOLEAN, OPEN_STRUCTURE_COM
    new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.BOOLEAN, SYNCHRONIZE_SCROLL
    new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.BOOLEAN, SHOW_PSEUDO_CONFLI
    new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.BOOLEAN, INITIALLY_SHOW_ANC
    new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.BOOLEAN, SHOW_MORE_INFO),
    new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.BOOLEAN, IGNORE_WHITESPACE)
    new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.BOOLEAN, PREF_SAVE_ALL_EDIT

    new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.BOOLEAN, NEW_PREFERENCE),

    new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.STRING, AbstractTextEditor.
    new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.BOOLEAN, AbstractTextEditor
    //new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.BOOLEAN, USE_SPLINES),
    new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.BOOLEAN, USE_SINGLE_LINE),
    //new OverlayPreferenceStore.OverlayKey(OverlayPreferenceStore.BOOLEAN, USE_RESOLVE_UI),
};


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);
    //store.setDefault(USE_SPLINES, false);
    store.setDefault(USE_SINGLE_LINE, true);
```

Symbol	File	Support	Confidence
initDefaults(IPreferenceStore store)	ComparePreferencePage.java	8	1.0
org.eclipse.compare/plugin.properties	plugin.properties	7	0.875
org.eclipse.compare/buildnotes_compare.html	buildnotes_compare.html	6	0.75
TextMergeViewer(Composite parent, int style, CompareConfiguration configuration)	TextMergeViewer.java	6	0.75
propertyChange(PropertyChangeEvent event)	TextMergeViewer.java	6	0.75
createGeneralPage(Composite parent)	ComparePreferencePage.java	5	0.625
createTextComparePage(Composite parent)	ComparePreferencePage.java	5	0.625
handleDispose(DisposeEvent event)	TextMergeViewer.java	4	0.5

Figure 2.1: After the programmer has made some changes to the ECLIPSE source (above), ROSE suggests locations (below) where, in similar transactions in the past, further changes were made.

programmer with the same rationale in a short time window. The ROSE server then stores the transactions in a database. This representation is independent from the concrete version system used and can be used for arbitrary analyses. A unique feature of ROSE is that it maps the changes to *syntactic entities* such as methods, attributes, or sections [1]. ROSE is thus able to detect coupling at a much finer granularity than, say, files or directories.

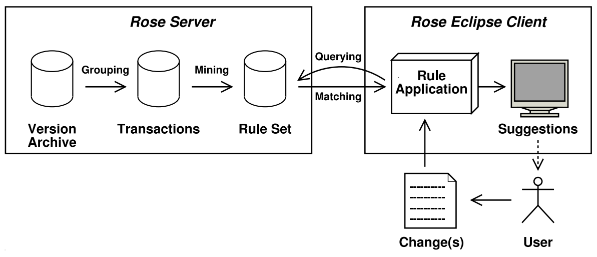


Figure 2.2: The data flow through ROSE.

The ROSE client makes the database accessible to programmers: As soon as the programmer makes a change, ROSE mines the database for possible related changes and presents these to the programmer in a list at the bottom of the screen (Figure 2.1). This is a very efficient process, taking at most 0.5 seconds; the programmer can examine these suggested locations simply by clicking on them.

Do ROSE's recommendations make sense? Yes and no. ROSE is not able to predict every single change. An evaluation on eight large open source projects [2] shows a recall of only about 15%, meaning that only a sixth of the actually changed entities could be predicted by ROSE. On the other hand, the recommendations have a high likelihood to be correct: the topmost three suggestions contain a correct location with a likelihood of 64%. Thus, if ROSE suggests something, it had better be taken seriously.

2.3 Some Perspectives

Our work with ROSE opens interesting new research perspectives in program analysis. Traditionally, program analysis has been concerned with source code alone. Leveraging the development history of the product obviously allows to reveal coupling that would otherwise be inaccessible to source code analysis—because we can detect coupling between programs and documentation, or between entities that are not even programs.

Yet, we have only begun to scratch the surface of what may be hidden in version archives and other process artifacts. In the future, we plan to exploit log messages and problem databases (which often are synchronized with changes); mailing lists or developer forums may be other sources of gathering knowledge. Of course, all of this information is fuzzy and insecure, especially when compared with the hard facts that source code analysis can extract. On the other hand, when it comes to understanding a program, a good hint may be better than no hint at all—and we're afraid that most of our programs are of such complexity that any hint may be precious.

More information about this and related work can be found on our web site

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

Bibliography

[1] Thomas Zimmermann and Peter Weißgerber. Preprocessing CVS data for fine-grained analysis. In *Proc. Intl. Workshop on Mining Software Repositories (MSR)*, Edinburgh, Scotland, May 2004.

[2] Thomas Zimmermann, Peter Weißgerber, Stephan Diehl, and Andreas Zeller. Mining version histories to guide software changes. In *Proc. 26th International Conference on Software Engineering (ICSE)*, Edinburgh, Scotland, May 2004.