2IS55 Software Evolution

Program differencing

Alexander Serebrenik

TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

Assignment 4: Reminder

- Deadline: March 29
  - One week left!
  - If you do not like Java, take a different language
    - As long as it is supported by you duplication detector
  - Releases and not revisions!
- Questions?

- Assignment 5:
  - Published on March 29
  - Deadline: April 19

Sources

"Program differencing"

Miryung Kim

Recap: Program differencing

- Input: Two programs

- Output:
  - Differences between the two programs
  - Unchanged code fragments in the old version and their corresponding locations in the new version

- Last week:
  - Text-based (diff)
  - AST-diff [Yang 1992]
  - Enhanced control flow graph [Apiwattanapong et al. 2007]

Changes never come alone

- Search and replace
- Check-in comment: "Common methods go in an abstract class. Easier to extend/maintain/fix"

- Change: a rule rather than a set application results
  - Rules can have exceptions

- Idea [Kim and Notkin 2009]
  - Observe differences between subsequent versions
  - Formalize them as facts
  - Discover rules (à la data mining)
  - Record exceptions

Structural changes [Kim and Notkin 2009]

- Program ⇒ collection of facts

v_1

```
class Bus {
   void start(Key c) {
      c.on = false; }
}

class Key {
   boolean on = false;
   void chk (Key c) {…}
   void out() {…}
}
```

FB_0

```
type("Bus")
method("Bus.start","start","Bus")
access("Key.on","Bus.start")
type("Key")
field("Key.on", "on", "Key")
method("Key.chk","chk","Key")
method("Key.out","out","Key")
```

v_2

```
class Bus {
   void start(Key c) {
      log(); }
}

class Key {
   boolean on = false;
   void chk (Key c) {…}
   void output() {…}
}
```

FB_n

```
type("Bus")
method("Bus.start","start","Bus")
calls("Bus.start","log")
type("Key")
field("Key.on", "on", "Key")
method("Key.chk","chk","Key")
method("Key.output","output","Key")
```

Structural changes [Kim and Notkin 2009]

- **Calculate the differences**

type("Bus")
method("Bus.start","start","Bus")
access("Key.on","Bus.start")
type("Key")
field("Key.on", "on", "Key")
method("Key.chk","chk","Key")
method("Key.out","out","Key")

type("Bus")
method("Bus.start","start","Bus")
calls("Bus.start","log")
type("Key")
field("Key.on", "on", "Key")
method("Key.chk","chk","Key")
method("Key.output","output","Key")

ΔFB

deleted_access("Key.on","Bus.start")
added_calls("Bus.start","log")
~~deleted_method("Key.out","out","Key")~~
~~added_method("Key.output","output","Key")~~

Deleting Key.out and adding Key.output can be identified as renaming and removed.

Learn the rules

- **Rule inference should depend on ΔFB**
  - **Is this enough? Why?**
  - **Context information is lost!**
    - **We need FB_0 and FB_n**
    - **To distinguish between the facts: past_ and current_**

past_type("Bus")
past_method("Bus.start","start","Bus")
past_access("Key.on","Bus.start")
past_type("Key")
past_field("Key.on", "on", "Key")
past_method("Key.chk","chk","Key")
past_method("Key.out","out","Key")

current_type("Bus")
current_method("Bus.start","start","Bus")
current_calls("Bus.start","log")
current_type("Key")
current_field("Key.on", "on", "Key")
current_method("Key.chk","chk","Key")
current_method("Key.output","output","Key")

deleted_access("Key.on","Bus.start")
added_calls("Bus.start","log")

Rules

- **Datalog style**
- **Restrictions:**
  - **Antecedent: one type of facts**
  - **Consequent: only deleted_ or added_**

past_*	⇒	deleted_*	feature or dependency removal
past_*	⇒	added_*	consistent clone update
current_*	⇒	added_*	feature addition
deleted_*	⇒	added_*	API migration
added_*	⇒	deleted_*	

How are the rules learned?

Algorithm 1: LSdiff Rule Inference Algorithm
Input: FB_o, FB_n, ΔFB, m, a, k, and β
Output: L and U
/* Initialize R, a set of ungrounded rules; L, a set of learned rules; and U, a set of facts in ΔFB that are not covered by L. */
R := ∅, L := ∅, U := ΔFB;
U := applyDefaultWinnowingRules (ΔFB, FB_o, FB_n); /* reduce ΔFB with default winnowing rules. */
R := createInitialRules (m); /* create rules with an empty antecedent by enumerating all possible consequents. */

- **Winnowing rules remove trivial dependencies**
  - **"if a class is deleted, so are all its methods"**
- **Ungrounded – with variables**

U

deleted_access("Key.on","Bus.start")
added_calls("Bus.start","log")

R

deleted_access(f,m)
added_calls(m1,m2)

How are the rules learned?

Algorithm 1: LSdiff Rule Inference Algorithm
Input: FB_o, FB_n, ΔFB, m, a, k, and β
Output: L and U
/* Initialize R, a set of ungrounded rules; L, a set of learned rules; and U, a set of facts in ΔFB that are not covered by L. */
R := ∅, L := ∅, U := ΔFB;
U := applyDefaultWinnowingRules (ΔFB, FB_o, FB_n); /* reduce ΔFB with default winnowing rules. */
R := createInitialRules (m); /* create rules with an empty antecedent by enumerating all possible consequents. */
foreach $i = 1 \ldots k$ **do**
  R := extendUngroundedRules (R) ; /* extend all ungrounded rules in R by adding all possible literals to their antecedent. */

- **Winnowing rules remove trivial dependencies**
  - **"if a class is deleted, so are all its methods"**
- **Ungrounded – with variables**

R

deleted_access(f,m) ⇐
  past_method(m, mclass, mshort)
deleted_access(f,m) ⇐
  past_field(f, fshort, fclass)
deleted_access(f,m) ⇐
  past_access(f,m)

How are the rules learned?

Algorithm 1: LSdiff Rule Inference Algorithm
Input: FB_o, FB_n, ΔFB, m, a, k, and β
Output: L and U
/* Initialize R, a set of ungrounded rules; L, a set of learned rules; and U, a set of facts in ΔFB that are not covered by L. */
R := ∅, L := ∅, U := ΔFB;
U := applyDefaultWinnowingRules (ΔFB, FB_o, FB_n); /* reduce ΔFB with default winnowing rules. */
R := createInitialRules (m); /* create rules with an empty antecedent by enumerating all possible consequents. */
foreach $i = 1 \ldots k$ **do**
  R := extendUngroundedRules (R) ; /* extend all ungrounded rules in R by adding all possible literals to their antecedent. */
  foreach $r \in$ R **do**
    G := createPartiallyGroundedRules (r) ; /* try all possible constant substitutions for r's variable. */

R

deleted_access(…) ⇐ past_method(…)
deleted_access (…) ⇐ past_field(…)
deleted_access ("Key.on","Bus.start") ⇐
  past_access ("Key.on","Bus.start")
deleted_access ("Key.on","Key.chk") ⇐
  past_access ("Key.on","Key.chk")
deleted_access ("Key.on","Key.out") ⇐
  past_access ("Key.on","Key.out")
deleted_access ("Key.on",m) ⇐
  past_access ("Key.on",m)
deleted_access (f,"Bus.start") ⇐
  past_access (f,"Bus.start")
deleted_access (f,"Key.chk") ⇐
  past_access (f,"Key.chk")
deleted_access (f,"Key.out") ⇐
  past_access (f,"Key.out")
deleted_access(f,m) ⇐
  past_access(f,m)

How are the rules learned?

Algorithm 1: LSdiff Rule Inference Algorithm

Input: FB_o, FB_n, ΔFB, m, a, k, and β
Output: L and U
/* Initialize R, a set of ungrounded rules; L, a set of learned rules; and U, a set of facts in ΔFB that are not covered by L. */
R := ∅, L := ∅, U := ΔFB;
U := applyDefaultWinnowingRules (ΔFB, FB_o, FB_n); /* reduce ΔFB with default winnowing rules. */
R := createInitialRules (m); /* create rules with an empty antecedent by enumerating all possible consequents. */
foreach $i = 1 \ldots k$ do
  R := extendUngroundedRules (R) ; /* extend all ungrounded rules in R by adding all possible literals to their antecedent. */
  foreach $r \in$ R do
    G := createPartiallyGroundedRules (r) ; /* try all possible constant substitutions for r's variable. */
    foreach g in G do
      if isValid (g) then
        L :=L ∪ {g};
        U :=U − {g.matches};
      end
    end
  end
  R :=selectRules (R, β) ; /* select the best β rules in R */
end

- **Validity**
  - **≥ m matching facts**
  - **Match/(Match+NonMatch) ≥ a**

R (examples)
deleted_access ("Key.on","Bus.start")
⇐ past_access ("Key.on","Bus.start")
One fact, 100% precision
deleted_access ("Key.on","Key.chk") ⇐
past_access ("Key.on","Key.chk")
No matching facts
deleted_access ("Key.on",m) ⇐
past_access ("Key.on",m)
One fact, 100% precision
deleted_access (f,"Bus.start") ⇐
past_access (f,"Bus.start")
One fact, 100% precision
deleted_access(f,m) ⇐ past_access(f,m)
One fact, 100% precision

How are the rules learned?

Algorithm 1: LSdiff Rule Inference Algorithm

Input: FB_o, FB_n, ΔFB, m, a, k, and β
Output: L and U
/* Initialize R, a set of ungrounded rules; L, a set of learned rules; and U, a set of facts in ΔFB that are not covered by L. */
R := ∅, L := ∅, U := ΔFB;
U := applyDefaultWinnowingRules (ΔFB, FB_o, FB_n); /* reduce ΔFB with default winnowing rules. */
R := createInitialRules (m); /* create rules with an empty antecedent by enumerating all possible consequents. */
foreach $i = 1 \ldots k$ do
  R := extendUngroundedRules (R) ; /* extend all ungrounded rules in R by adding all possible literals to their antecedent. */
  foreach $r \in$ R do
    G := createPartiallyGroundedRules (r) ; /* try all possible constant substitutions for r's variable. */
    foreach g in G do
      if isValid (g) then
        L :=L ∪ {g};
        U :=U − {g.matches};
      end
    end
  end
  R :=selectRules (R, β) ; /* select the best β rules in R */
end

- **For more realistic examples**
  - **Thresholds do matter**
  - **More general = less accurate**
- **At the next step more antecedents are added, e.g.,**
  - **past_method(m,"start",t) ∧ past_subtype("Car",t) ⇒ added_calls(m,"Key.chk")**
- **β controls the number of rules kept for extension at the next iteration**

Evaluation

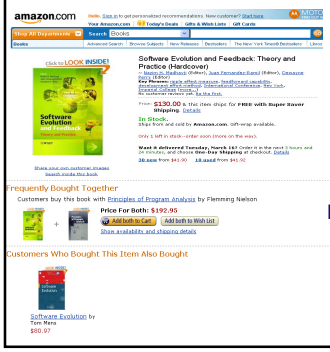
- **On average, 75% of facts in ΔFB are covered by inferred rules**
  - **~75% of structural differences are systematic change.**
- **Concise:**
  - **Textual diff: 997 lines, 16 files**
  - **LSdiff: 7 rules and 27 facts**

How did the users experience the tool?

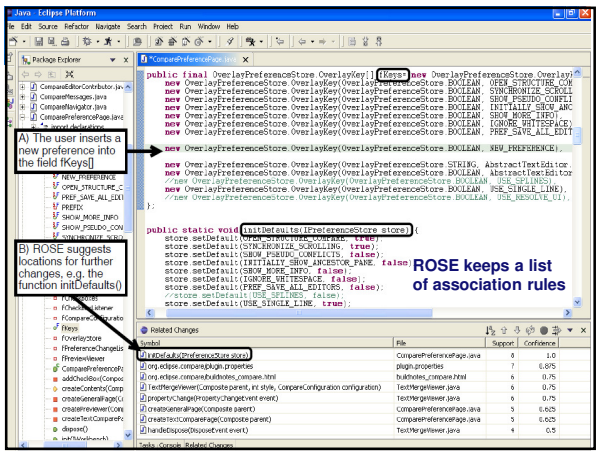
- **Positive**
  - **"You can't infer the intent of a programmer, but this is pretty close."**
  - **"This 'except' thing is great!"**
- **Negative**
  - **"This looks great for big architectural changes, but I wonder what it would give you if you had lots of random changes."**
  - **"This will look for relationships that do not exist."**

What about "random" changes?

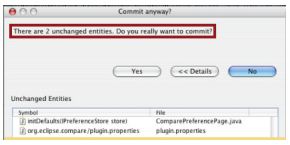
- **eROSE [Zimmermann, Weißgerber, Diehl, Zeller '04]**



Developers who modified this function also modified…

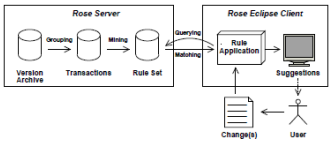


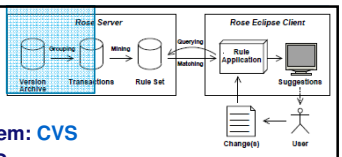
eROSE



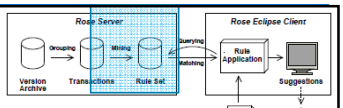
- ROSE alerts for incomplete changes

How?





- Version control System: CVS
- Shortcomings of CVS
  - Only file versioning rather than product versioning
    - Sliding window
  - Implicit merges
    - Goal: Related changes
    - Ignore merges – ignore changes of >30 files
  - Text-based version control
    - Analysis links software artefacts (functions, etc)
    - Parser!

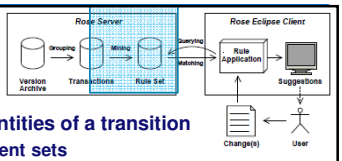



```
public static void initDefaults(IPreferenceStore store){
    store.setDefault(CREA_STRUCTURE_COMPARE, true);
    store.setDefault(SYNCHRONIZE_SCROLLING, true);
    store.setDefault(SHOW_PSEUDO_CONFLICTS, false);
    store.setDefault(INITIALLY_SHOW_ANCESTOR_PANE, false);
    store.setDefault(SHOW_MORE_INFO,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

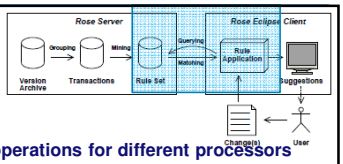
Related Changes

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

- Support |A∧B|
- Confidence |A∧B| : |A|



- Entity set = (some) entities of a transition
  - We need frequent event sets
  - Naïve:
    - All possible subsets
    - Select the frequent ones
  - Apriori algorithm
    - All subsets of a frequent entity set should be frequent
    - Start with frequent entities and extend them one by one
    - Rule: $(Y \setminus X) \Rightarrow X$ for any $X \subseteq Y$, Y is a frequent entity set
    - Pre-calculation of all the rules
  - eROSE
    - No pre-calculation and only one antecedent



- Examples of rules
  - Costs of Assembly operations for different processors
    - { (i386.h, *type, processor_cost)* } ⇒
      { (i386.c, *var, i386_cost), (i386.c, var, i486_cost),*
      (i386.c, *var, k6_cost), (i386.c, var, pentium_cost),*
      (i386.c, *var, pentiumpro_cost)* }
  - Cross-language
    - { (Qdmodule.c, *func, GrafObj getattr()*) } ⇒
      { (qdsupport.py, *func, outputGetattrHook()*) }
- Application of rules R to state Σ: { x | (Σ ⇒ x) ∈ R}

Experimental evaluation

- Recall: 0.15
  - suggestion included 15% of all changes that were carried out
- Precision: 0.26
  - 26% of all recommendations were correct
- Likelyhood:
  - 70% of all transactions, topmost three suggestions contain a changed entity.
- EROSE learns quickly
  - within 30 days
- Extensive evaluation

Conclusions so far

- Two approaches to identification of related differences:
  - Both based on data mining/rule learning
    - [Kim and Notkin] FOL rules
    - [EROSE] Association rules
  - Interesting ideas, not always impressive results
    - A lot of improvement is possible!