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

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

\[
\begin{align*}
\text{class Bus } & \{ \\
& \begin{align*}
& \text{void start(Key c) } \\
& \{ \\
& \quad \text{c.on = false;} \\
& \} \\
& \} \\
\end{align*} \\
\text{class Key } & \{ \\
& \begin{align*}
& \text{boolean on = false;} \\
& \text{void chk (Key c) […]} \\
& \text{void out()} […] \\
& \} \\
& \} \\
\end{align*} \\
\end{align*}
\]
Structural changes [Kim and Notkin 2009]

- Calculate the differences

```
| type("Bus") | method("Bus.start","start","Bus") |
| type("Key") | method("Key.on","on","Key") |
```

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

Learn the rules

- Rule inference should depend on \( \Delta FB \)
- Is this enough? Why?
- Context information is lost!
  - We need \( FB_0 \) and \( FB_n \)
  - To distinguish between the facts: past and current

Rules

- Datalog style
- Restrictions:
  - Antecedent: one type of facts
  - Consequent: only deleted or added

```
past_* \implies \text{deleted}_*
\quad \text{feature or dependency removal}

past_* \implies \text{added}_*
\quad \text{consistent clone update}

current_* \implies \text{added}_*
\quad \text{feature addition}

deleted_* \implies \text{added}_*
\quad \text{API migration}

added_* \implies \text{deleted}_*
```
How are the rules learned?

- 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,m) ⇐ past_access(f,m)
One fact, 100% precision

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) \rightarrow 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 ∆FB 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 merge
    - Goal: Related changes
      - Ignore merges – ignore changes of >30 files
    - Text-based version control
      - Analysis links software artefacts (functions, etc)
      - Parser!

- Rules (similar to Kim, Notkin but not identical)
  - Apriori algorithm
    - All subsets of a frequent entity set should be frequent
    - Start with frequent entities and extend them one by one
    - Rule: \( Y \subseteq X \) 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

- Entity set = (some) entities of a transition
  - We need frequent event sets
    - Naive:
      - 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: \( Y \subseteq X \) 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
  - \( \{ (i386.h, \text{type, processor\_cost}) \} \Rightarrow \{ (i386.c, \text{var, i386\_cost}), (i386.c, \text{var, i486\_cost}), (i386.c, \text{var, k6\_cost}), (i386.c, \text{var, pentium\_cost}), (i386.c, \text{var, pentiumpro\_cost}) \} \)
- Cross-language
  - \( \{ (\text{dmodule}, \text{func, GrafoObj getatrr}) \} \Rightarrow \{ (\text{dsupport.py, func, outputGetattrHook}) \} \)
- Application of rules \( R \) to state \( \Sigma: \{ x \mid (\Sigma \Rightarrow x) \in R \} \)

Experimental evaluation
- Recall: 0.15
  - suggestion included 15% of all changes that were carried out
- Precision: 0.26
  - 26% of all recommendations were correct
- Likelihood:
  - 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!