Assignment 3: Reminder

- Deadline: Tonight at midnight
- Assignment 4 (till March 29):
  - Code duplication
  - Replication study of a scientific paper
  - More details to come

Where are we now?

- Last week:
  - Code cloning, code duplication, redundancy...
  - Type 1, 2, 3, 4 clones (more refined classification possible)
  - Useful: reliability, reduced time, code preservation
  - Harmful: more interrelated code, more bugs
  - Ignore, eliminate, prevent, manage
  - Detection mechanisms
    - Text-based
    - Metrics-based
    - Token-based

Today

- Clone detection techniques
  - AST-based
    - [Baxter 1996]
    - AST+Tokens combined [Koschke et al. 2006]
  - Program Dependence Graph
    - [Krinke 2001]
  - Comparison of different techniques

AST-based clone detection [Baxter 1996]

- If we have a tokenizer we might also have a parser!
- Applicability: the program should be parseable

- Compare every subtree with every other subtree?
  - For an AST of n nodes: O(n³)
  - Similarly to text: Partitioning with a hash function
  - Works for Type 1 clones
AST-based detection

- Type 2
  - Either take a bad hash function ignoring small subtrees, e.g., names
  - Or replace identity by similarity
  
  $$\text{Similarity}(T_1, T_2) = 2 \times \frac{\text{Same}(T_1, T_2)}{2 \times \text{Same}(T_1, T_2) + \text{Difference}(T_1, T_2)}$$

- Type 3
  - Sequences of subtrees
  - Go from Type 2-cloned subtrees to their parents

- Rather precise but still slow

Recapitulation from the last week

- [Baker 1995]
  - Token-based
  - Very fast:
    - 1.1 MLOC, minimal clone size: 30 LOC
    - 7 minutes on SGI IRIX 4.1, 40MHz, 256 MB

- [Baxter 1996]
  - AST-based
  - Precise but slow

- Idea: Combine the two! [Koschke et al. 2006]
  - In fact they do not use [Baker 1995] but a different token-based approach

AST + Tokens [Koschke et al. 2006]

Code → AST → Serialized AST → Token clones

- Preorder
- Number = number of descendants

We do not distinguish between Type 1 and Type 2
**Incomplete syntactical units**

- "return id ; } int id( ) { int id ;"
- Correct as a clone but useless for reengineering
- Should be split:
  - return id; } int id() { int id;

```
int id;
// id
if (id)
    return id;
```

**AST + Tokens: Discussion**

- Splitting algorithm can be updated to deal with sequences
- Compromise of recall and precision
  - Recall: % of the true clones found
  - Precision: % of clones found that are true
- Reasonably fast (faster than pure AST)

**Next step**

- AST is a tree is a graph
- There are also other graph representations
  - Object Flow Graph (weeks 3 and 4)
  - UML class/package/... diagrams
  - Program Dependence Graph
- These representations do not depend on textual order
  - \( \{ x = 5; y = 7; \} \) vs. \( \{ y = 7; x = 5; \} \)

**[Krinko 2001] PDG based**

- Vertices:
  - entry points, in- and output parameters
  - assignments, control statements, function calls
  - variables, operators
- Edges:
  - immediate dependencies
  - value dependencies
  - reference dependencies
  - data dependencies
  - control dependencies
- Not in this example

- \( y = b + c \)
- \( x = y + z \)

**Identification of similar subgraphs – Theory**

- Start with 1 and 10
- Partition the incident edges based on their labels
- Select classes present in both graphs
- Add the target vertices to the set of reached vertices
- Repeat the process
- “Maximal similar subgraphs”
Identification of similar subgraphs – Practice

- Sorts of edges are labels
- We also need to compare labels of vertices
- We should stop after k iterations
  - Higher k ⇒ higher recall
  - Higher k ⇒ higher execution time
  - Experiment: k = 20

Clone detection techniques – what have we seen?

- Text-based
  - [Ducasse et al. 1999, Marcus and Maletic 2001]
- Metrics-based
  - [Mayrand et al. 1996]
- Token-based
  - [Baker 1995, Kamiya et al. 2002]
- AST-based
  - [Baxter 1996]
- AST+Tokens combined [Koschke et al. 2006]
- Program Dependence Graph
  - [Krinke 2001]

Choosing your tools: Precision / Recall

- Quality depends on scenario [Type 1, Type 2, Type 3]
- [Roy, Cordy, Koschke, Science of Comp Progr 2009]
- Brief summary, 6 is maximal grade, 0 – minimal

<table>
<thead>
<tr>
<th>Tool</th>
<th>Quality</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duploc</td>
<td>4</td>
<td>0</td>
<td>2.8</td>
</tr>
<tr>
<td>Marcus and Maletic</td>
<td>2.6</td>
<td>1.8</td>
<td>1.6</td>
</tr>
<tr>
<td>Dude</td>
<td>4.6</td>
<td>0</td>
<td>4.4</td>
</tr>
<tr>
<td>Simian</td>
<td>4.3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Dup</td>
<td>4</td>
<td>2.8</td>
<td>0</td>
</tr>
<tr>
<td>CCFinder</td>
<td>5</td>
<td>3.8</td>
<td>0.8</td>
</tr>
<tr>
<td>CloneDr</td>
<td>6</td>
<td>4.3</td>
<td>3.8</td>
</tr>
<tr>
<td>cpdetector</td>
<td>6</td>
<td>3.8</td>
<td>0</td>
</tr>
<tr>
<td>Mayrand</td>
<td>3.3</td>
<td>4.8</td>
<td>3.4</td>
</tr>
<tr>
<td>Duplix</td>
<td>5</td>
<td>4.8</td>
<td>4</td>
</tr>
</tbody>
</table>

Which technique/tool is the best one?

- Quality
  - Precision
  - Recall
- Usage
  - Availability
  - Dependence on a platform
  - Dependence on an external component (lexer, tokenizer, …)
- Input/output format
  - Programming language
  - Clones
  - Granularity
  - Types
  - Pairs vs. groups
- Technique
  - Normalization
  - Storage
  - Worst-case complexity
  - Pre-/postprocessing
  - Validation
  - Extra: metrics

Code duplication metrics

- Measuring allows
  - Reporting to the business level
  - Comparing two systems
  - Comparing two versions of the same system
- Clone duplication metrics
  - Per system
    - % duplicated lines
  - Per file
    - Usually will contain clones from different groups
  - Per clone group
    - Usually will contain fragments from different files
- Code duplication: File metrics
  - Given a file F:
    - NBR (neighbor):
      - Count of files containing a cloned fragment with F
    - RSA (ratio of similarity between another files):
      - % tokens covered by a clone between F and another file.
    - RSI (ratio of similarity within the file):
      - % tokens covered by a clone in F.
    - CVR (coverage):
      - % tokens covered by any code clone.
      - max(RSA, RSI) <= CVR <= RSA+RSI
Code duplication: Clone group metrics

- POP (population):
  - Count of code fragments of the code clone
- NIF:
  - Count of source files that include one or more code fragments of the code clone. By definition, NIF ≤ POP
- RAD (radius):
  - Range of the source code fragments of a code clone in the directory hierarchy.

Clone detection - conclusions

- Many different techniques
  - Text, metrics, tokens, AST, graphs
- Advantages and disadvantages
  - Precision vs. recall
  - Suitability for the task

Assignment 4: Code clone coverage of an evolving system

- Measure % clone coverage between subsequent versions of a system
- Visualize using the heat map
- Discuss the patterns observed
- More details on Peach

New Topic: Mining Software Repositories

- Evolution is reflected in
  - version control systems
  - code
  - documentation
  - bug trackers
  - mail archives and forums
  - wiki
  - documentation
  - discussion
  - Twitter
  - ...

Mining Software Repositories

- Goals
  - Understanding evolution
  - Predicting future phenomena
- Important topic
  - Series of lectures on different aspects
  - Today and on March 29 - program differencing
  - March 29 – version control systems (invited lecture)

Program differencing

- Why?
  - Version control
    - What have we changed since...
    - How can we merge the changes by Alice and Bob?
  - What do we need to retest?
  - What is the impact of our change?
- Formally
  - Input: Two programs
  - Output:
    - Differences between the two programs
    - Unchanged code fragments in the old version and their corresponding locations in the new
- Similar to clone detection
  - Comparison of lines, tokens, trees and graphs
**Diff: Longest common subsequence**

- Program: sequence of lines
- Object of comparison: line

**Comparison:**
- 1:1
- lines are identical
- matched pairs cannot overlap

**Technique:** longest common subsequence
- Minimal number of additions/deletions steps
- Dynamic programming

---

**Longest common subsequence**

- Programs X (n lines), Y (m lines)
- Data structure $C[0..n,0..m]$
- Init: $C(r,0)=0$, $C(0,c)=0$ for any $r$ and $c$

$$
<table>
<thead>
<tr>
<th>\text{p0 mA()}</th>
<th>\text{p1 if (pred_a)}</th>
<th>\text{c0 mA()}</th>
<th>\text{c1 if (pred_a)}</th>
<th>\text{c2 if (pred_a)}</th>
<th>\text{c3 if (pred_a)}</th>
<th>\text{c4 if (pred_a)}</th>
<th>\text{c5 c6}</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>diff:r=0</td>
<td>if X[r]=Y[c] then C[r,c]=C[r-1,c-1]+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>else C[r,c]=max(C[r-1,c],C[r,c-1])</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Longest common subsequence**

- For every $r$ and every $c$
  - If $X[r]=Y[c]$ then $C[r,c]=C[r-1,c-1]+1$
  - Else $C[r,c]=\max(C[r-1,c],C[r,c-1])$

- Start with $r=m$ and $c=n$
- backTrace($r,c$)
  - If $r=0$ or $c=0$ then 
  - If $X[r]=Y[c]$ then backTrace($r-1,c-1$)+X[$r$]
  - Else 
    - If $C[r,c-1] > C[r-1,c]$ then backTrace($r,c-1$) else backTrace($r-1,c$)

---

**Diff: Summarizing**

- Comparison:
  - 1:1, identical lines, non-overlapping pairs
- Technique: longest common subsequence
- What kind of code modifications will diff miss?
  - Copy & paste: apple \Rightarrow apple
    - 1:1 is violated
  - Move: apple \Rightarrow aple
Construct ASTs for the input programs

Recursive algo pairwise subtree comparison

\[ M[i,j] = \max(M[i,j-1], \] 
\[ M[i-1,j], W[i,j]) \] 

When the computation is finished, return \( M[n,m]+1 \)

If root symbols differ return \( 0 \)

When the computation is finished, return \( M[n,m]+1 \)

If root symbols differ return \( 0 \)
Continuing the process

Advantages
- Respect the parent-child relations
- Ignore the order between siblings

Disadvantages
- Sensitive to tree level changes ("if (pa")
- Ignore dependencies such as data flow, etc

Comparison [Apiwattanapong et al. 2007]

- Compare P and P', determine matching classes and interfaces, add the pairs to C or I
  - matching = fully qualified names coincide
  - No match = record as added/removed class or interface

- For every pair (m, m') of methods in M
  - Build enhanced control flow graphs for m and m'
  - Enhanced control flow graph
  - Traditional "control flow graph" + OO-specific enhancements

Even more about the dataflow

- What part of the behaviour is affected by the change?
  - Different m1() is called
  - Different catch block is relevant

- For every pair in C or I record matching methods in M
  - matching = fully qualified names coincide
  - No match = record as added/removed method

Comparison [Apiwattanapong et al. 2007]

- For every pair in C or I record matching methods in M
  - matching = fully qualified names coincide
  - No match = record as added/removed method

For every pair (m, m') of methods in M
- Build enhanced control flow graphs for m and m'
- Enhanced control flow graph
- Traditional "control flow graph" + OO-specific enhancements

ECFG

For every pair (m, m') of methods in M
- Build enhanced control flow graphs for m and m'
- Enhanced control flow graph
- Traditional "control flow graph" + OO-specific enhancements
How can we compare ECFGs?

- Structure preservation
- Simplify each one of the ECFGs using reduction rules

Hammock

- Recursively compare the simplified versions
  - Individual nodes: labels should be the same
  - Aggregated nodes: % matched individuals > threshold
  - No match found: try to match using lookahead
  - Allows matching at different aggregation levels
  - Record whether the nodes have been matched
    - (c,c,"unchanged") vs. (c,c,"modified")

Evaluation

<table>
<thead>
<tr>
<th>Pair</th>
<th>Number of Nodes</th>
<th>Avg. Covered</th>
<th>Avg. Correct/ Test Case</th>
<th>Correct/ Test Suite</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1,v2</td>
<td>22006</td>
<td>12591.75</td>
<td>98.57</td>
<td>97.17</td>
</tr>
<tr>
<td>v1,v3</td>
<td>22006</td>
<td>12620.75</td>
<td>98.46</td>
<td>97.17</td>
</tr>
<tr>
<td>v1,v4</td>
<td>22315</td>
<td>12661.23</td>
<td>98.03</td>
<td>96.20</td>
</tr>
</tbody>
</table>

- Number of nodes in v_i
- Avg. covered – average # nodes covered by a test case
- Avg. correct / Test case – # nodes correctly estimated as being covered by a test case, averaged over all test cases
- Correct / Test suite – for entire suite

Evaluation: Retesting

- Given:
  - Code: v_1, v_2, v_3, v_4 and a test suite for v_1 (~60% coverage)

- Process
  - Run T on v_1, collect nodes covered by T
  - Calculate differences v_i ↔ v_i and estimate coverage
    - (n,n',"unch.") ⇒ n' is covered by the same test case as n
    - (n,n',"modified")
    - Go for the predecessors q' of n until
      - Either q' is a branch⇒ n' is not covered
      - Or (q,q',"unch") for some q ⇒ n' is covered by all test cases covering q
  - Run T on v_i to check whether the estimates are correct

Evaluation

- Number of nodes in v_i
- Avg. covered
- Avg. correct / Test case
- Correct / Test suite

JDiff - summary

- Quality of estimates decreases if many changes become larger
- Two parameters: lookahead and threshold
  - Threshold = constant ⇒ larger lookahead = more time
  - Lookahead = constant ⇒
    - Threshold = 0 much faster than Threshold > 0
    - The actual value of threshold is of no importance
- CFG matching with special OO extensions
- Flexible (parameterized)
- Efficient and precise

Conclusion

- Today:
  - Advanced code cloning techniques (AST, graphs)
  - Code cloning metrics
  - Mining software repositories – setting up the scene
  - Program differencing techniques (text, AST, graphs)

- Next week
  - Advanced differing techniques (groups)
  - Invited talk: Organization of version control systems