Elmar Juergens, Florian Deissenboeck, Benjamin Hummel

Code Similarities
Beyond Copy & Paste

March 17th, 2010
14th CSMR, Madrid

Problem

Code Duplication

- Duplicated code hinders software maintenance
  - Changes often have to be performed in multiple locations
  - Unintentionally inconsistent modifications can introduce faults
- Clone detection tools help detect and manage cloned code
- Duplication is typically created by copy & paste

Beyond Copy & Paste

- Maintenance problems caused by semantic coupling (not by C&P)
- Coupling also for behaviorally similar code w/o copy & paste

Can clone detection find behaviorally similar code of ind. origin?

Agenda

Notions of Similarity

Research Questions

Empirical Study

Conclusion

Notions of Similarity

Representational: Clone

- Similarity relation defined on (normalized) program representation
- Actual program representation and similarity relation varies

```
// Das zuletzt abgerechnete Jahr herausfinden (dieses darf editiert werden!)
for (int jahr = BasisdatenUtil.AktuellesGeschaeftsjahr; jahr >= BasisdatenUtil.Aktuelles
{
  // Nur Jahre bis zum St
  if ( jahr < stand.Range
  {
    break;
  }

  if ( BasisdatenUtil.Abge
  {
    if ( zuletztAbgerechn
    {
      vorZuletztAbgerec
    }
    else
    {
      zuletztAbgerechne
    }
  }
}
```

```
// Das zuletzt abgerechnete Jahr herausfinden (dieses darf editiert werden!)
for (int jahr = BasisdatenUtil.AktuellesGeschaeftsjahr; jahr >= BasisdatenUtil.Aktuelles
{
  // Nur Jahre bis zum Stand Begin betrachten, da weitere Jahre aus der betrachtung
  if ( jahr < stand.Range.RealBegin.Year )
  {
    break;
  }

  if ( BasisdatenUtil.Abgerechnet(jahr) )
  {
    if ( zuletztAbgerechnetesJahr > -1 )
    {
      vorZuletztAbgerechnetesJahr = jahr;
    }
    else
    {
      zuletztAbgerechnetesJahr = jahr;
    }
  }
}
```

Behavioral: Simion

- Similarity relation defined as function on input / output variables
- Behaviorally similar code fragment
- Different similarities for diff. use cases

```
int x, y, z;
```

```
z = x*y;
```

```
z = 0;
while (x > 0) {
  z += y;
  x -= 1;
}
while (x < 0) {
  z -= y;
  x += 1;
}
```

Research Questions

RQ1: How successfully can existing clone detection tools detect simions that do not result from copy & paste?

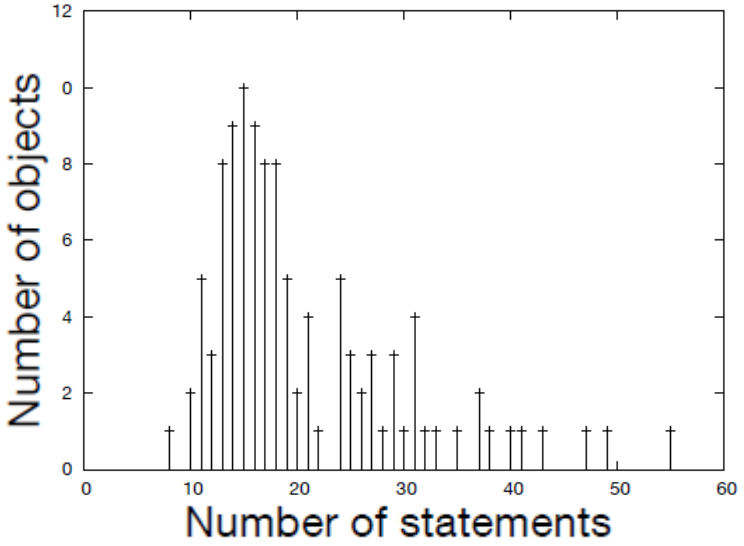
RQ2: Is program-representation-based clone detection in principle suited to detect simions that do not result from copy & paste?

RQ3: Do simions that do not result from copy & paste occur in practice?

RQ1&2: Study Objects

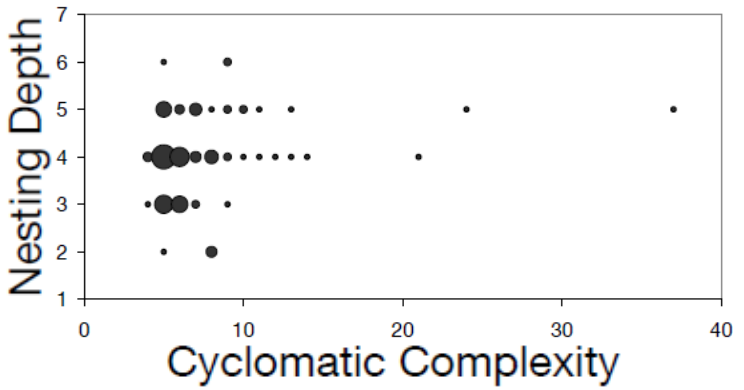
Creation Process

- Specification of simple email address validator function
- 400 cs students asked to implement spec in Java
- Voluntary & anonymous
- Supervision to avoid copy & paste



Resulting Study Objects

- 109 implementations with similar behavior (pass test suite): simions
- Significant structural variation



RQ1: Design & Results

- Ideal detector: detects cloning relation btw. all pairs of study objects
- Recall as ratio of detected w.r.t. all clone pairs
- *Full clone recall*: complete clones; *partial clone recall*: any clone
- ConQAT (token-based) and DECKARD (AST-based) for detection

Detector	*Clone Types*	*Partial Recall*	*Full Recall*
ConQAT	1	0.4%	0.0%
ConQAT	1 & 2	2.3%	0.0%
ConQAT	1 & 2 & 3	3.2%	0.1%
DECKARD	1 & 2	5.1%	0.1%
DECKARD	1 & 2 & 3	9.7%	0.8%

⇒ **Existing clone detectors not well suited for simion detection**

RQ2: Design and Procedure

- Collect differences between study objects
- Categorize differences according to compensability
  (based on capabilities of existing approaches)

- Manually inspect and classify differences between pairs of study obj.
- 55 random pairs inspected; all study objects covered
- Program variation categorizations from [Metz00] and [Wills93] as starting point

Measure: ratio of pairs that only contain compensatable differences w.r.t. all inspected pairs

Program Variation Example

```
1  public String[] validateEmailAddresses(String
       addresses, char separator, Set<String>
       invalidAddresses) {

3    if (addresses == null)
4      return new String[0];
5    if (addresses.equals(""))
6      return new String[0];

8    List<String> valid = new ArrayList<String>();

10   String sep = String.valueOf(separator);
11   if (separator == '\\')
12     sep = "\\\\";
13   String[] result1 = addresses.split(sep);
14   System.out.println(Arrays.toString(result1));

16   for (String adr : result1) {
17     Matcher m = emailPattern.matcher(adr);
18     boolean ergebnis = m.matches();
19     if (ergebnis)
20       valid.add(adr);
21     else
22       invalidAddresses.add(adr);
23   }

25   return valid.toArray(new String[0]);
26 }
```

```
1  public String[] validateEmailAddresses(String
       addresses, char separator, Set<String>
       invalidAddresses) {

3    if(addresses == null || addresses.equals("")) {
4      return new String[]{}; }

6    else {
7      addresses.replace(" ", "");
8      ArrayList<String> validAddresses = new
         ArrayList<String>();

10     StringTokenizer tokenizer = new
         StringTokenizer(addresses, String.valueOf(
         separator));

12     while(tokenizer.hasMoreTokens()) {
13       String i = tokenizer.nextToken();
14       if(this.emailPattern.matcher(i).matches()){
15         validAddresses.add(i);
16       } else {
17         invalidAddresses.add(i);
18       }
19     }

21     return validAddresses.toArray(new String[]{});
22   }
23 }
```

Syntactic Variation

Organization Variation

Generalization

Unnecessary Code

Different algorithm or data structure

Results

- 6 program variation categories encountered in study objects
- Compensatable, if within reach of existing approach
- 5 classified as compensatable, 1 classified as non-compensatable
- Of 55 inspected clone pairs, only 4 had only compensatable differences

Program Variation	
Syntactic variation	
Organization variation	
Generalization	
Delocalization	
Unnecessary code	
Different algorithm or data structure	

Only 7% detectable by ideal program-representation based approaches

=> Existing approaches not well suited for simion detection

RQ3: Simions in Real World Software

Study Object

- Open source Java reference manager JabRef
- Apache Commons

Design & Procedure

- Pair-review of JabRef source code (Lim. to 6 kLoC of utility methods)

Results

- Many simions discovered; both inside JabRef and with Ap. Commons.
- Example: Class *net.sf.jabref.Util* (2.7 kLoC):
  - 32 of 52 gen. purp. methods (partial) simions
  - 11 behaviorally equivalent with methods from Apache commons
- Several faults encountered by comparison with library code.

Threats to Validity

Construct

- RQ1: Parameters influence detection results

  Very aggressive detection (doesn't break result validity)

Internal

- RQ2: Misclassification can impact results

  Conservative classific.: Independent of completeness or complexity
- RQ2: Program variation among study objects possibly reduced by:
  - all students had access to same test suite
  - signature of validator function (incl. types) was specified
  - Teams could ask tutors for help.

  But: These factors only increase chances to detect simions

Threats to Validity (2)

External

- RQ1: Only 2 clone detectors evaluated; others could be better

  Likely similar results: significant structural diversity of study objects

  Result of RQ2 upper bound for all existing tools.

- RQ2: Transferability for conclusions, not actual numbers

  But: expected variation greater for real world software (different practices and conventions across different teams and domains)

- RQ3: Transferability unclear; JabRef study only provides indication

  e.g. too early to reason about defect proneness of missed reuse opp.

Conclusion

Summary

- Existing clone detection tools unsuited to detect simions
- Program-representation-based similarity unsuited in general
- Indication that
  - Real world software contains simions
  - Reinvention more costly and error-prone than reuse

Future Work

- Novel detection approaches. Examples: [Rob09], [Jiang09]
- Dynamic Clone Detection: Google Research Grant
- Study replication (RQ1 & 2 during cs labs, RQ3 on open source SW)

Thank You!