Software Code Clone

Eduardo Figueiredo

http://www.dcc.ufmg.br/~figueiredo

Code Clone

- Code Clone, also called Duplicated Code, is a well known code smell in software systems
  - Code clones are often the result of copy & paste practices

- Studies show that about 5% to 20% of a software code is duplicated

Drawbacks of Code Clone

- Increase the probability of bug propagation (to clones)
- Increase probability of bad design, such as lack of inheritance
- Increase the cost of software maintenance
  - Several places to change instead of one

Code Clone Types I and II

- Type I: Identical code fragments except for variations in whitespace, layout, and comments
- Type II: Structurally identical fragments except for variations in identifiers, literals, types, layout, and comments

Examples of Types I and II

Type I

```
if (a>=b) {
   // Comment1'
   c=d+b;
   d=d+1;}
else // Comment2'
   c=d-a;
```

```
if (a>=b)
{ // Comment1''
   c=d+b;
   d=d+1;
}
else // Comment2''
   c=d-a;
```

Type II

```
if (a >= b) {
   c = d + b; // Comment1
   d = d + 1;}
else
   c = d - a; //Comment2
```

```
if (m >= n)
{ // Comment1'
   y = x + n;
   x = x + 5; //Comment3
}
else
   y = x - m; //Comment2'
```

Code Clone Types III and IV

- Type III: Copied fragments with further modifications
  - Statements can be changed, added, or removed in addition to variations in identifiers, literals, types, and comments
- Type IV: Fragments that perform the same computation, but implemented through different syntactic variants

Examples of Types III and IV

Type III

```
if (a >= b) {
  c = d + b; // Comment1
  d = d + 1;}
else
  c = d - a; //Comment2
```

```
if (a >= b) {
  c = d + b; // Comment1
  e = 1; // Added statement
  d = d + 1; }
else
  c = d - a; //Comment2
```

Type IV

```
int i, j=1;
for (i=1; i<=VALUE; i++)
  j=j*i;
```

```
int factorial(int n) {
  if (n == 0) return 1;
  else
    return n * factorial(n-1);
}
```

Another Example (Java API)

```
public int getSoLinger() throws SocketException{
  Object o = impl.getOption( SocketOptions.SO_LINGER );
  if (o instanceof Integer) {
    return((Integer) o).intValue();
  }
  else return -1;
}
```

Fragment I

Fragment II

```
public synchronized int getSoTimeout()
    throws SocketException{
  Object o = impl.getOption( SocketOptions.SO_TIMEOUT );
  if (o instanceof Integer) {
    return((Integer) o).intValue();
  }
  else return 0;
}
```

Similarities: Fragments I and II

Fragment I

```
public int getSoLinger() throws SocketException{
  Object o = impl.getOption( SocketOptions.SO_LINGER );
  if (o instanceof Integer) {
    return((Integer) o).intValue();
  }
  else return -1;
}
```

Exact Copy

Fragment II

```
public synchronized int getSoTimeout()
    throws SocketException{
  Object o = impl.getOption( SocketOptions.SO_TIMEOUT );
  if (o instanceof Integer) {
    return((Integer) o).intValue();
  }
  else return 0;
}
```

Differences: Fragments I and II

```
public int getSoLinger() throws SocketException{
  Object o = impl.getOption( SocketOptions.SO_LINGER );
  if (o instanceof Integer) {
    return((Integer) o).intValue();
  }
  else return -1;
}
```

Fragment I

Insertion

Fragment II

```
public synchronized int getSoTimeout()
    throws SocketException{
  Object o = impl.getOption( SocketOptions.SO_TIMEOUT );
  if (o instanceof Integer) {
    return((Integer) o).intValue();
  }
  else return 0;
}
```

Differences: Fragments I and II

Fragment I

```
public int getSoLinger() throws SocketException{
  Object o = impl.getOption( SocketOptions.SO_LINGER );
  if (o instanceof Integer) {
    return((Integer) o).intValue();
  }
  else return -1;
}
```

Replacement

Fragment II

```
public synchronized int getSoTimeout()
    throws SocketException{
  Object o = impl.getOption( SocketOptions.SO_TIMEOUT );
  if (o instanceof Integer) {
    return((Integer) o).intValue();
  }
  else return 0;
}
```

Differences: Fragments I and II

Fragment I

```
public int getSoLinger() throws SocketException{
  Object o = impl.getOption( SocketOptions.SO_LINGER );
  if (o instanceof Integer) {
    return((Integer) o).intValue();
  }
  else return -1;
}
```

Replacement

Fragment II

```
public synchronized int getSoTimeout()
    throws SocketException{
  Object o = impl.getOption( SocketOptions.SO_TIMEOUT );
  if (o instanceof Integer) {
    return((Integer) o).intValue();
  }
  else return 0;
}
```

Differences: Fragments I and II

Fragment I

```
public int getSoLinger() throws SocketException{
  Object o = impl.getOption( SocketOptions.SO_LINGER );
  if (o instanceof Integer) {
    return((Integer) o).intValue();
  }
  else return -1;
}
```

Replacement

Fragment II

```
public synchronized int getSoTimeout()
    throws SocketException{
  Object o = impl.getOption( SocketOptions.SO_TIMEOUT );
  if (o instanceof Integer) {
    return((Integer) o).intValue();
  }
  else return 0;
}
```

Bibliography

- M. Balazinska, E. Merlo, M. Dagenais, B. Lague, K. Kontogiannis. “Measuring Clone Based Reengineering Opportunities”. In Proceedings of the 6th International Software Metrics Symposium (METRICS), pp. 292-303, 1999.
- R. Koschke, R. Falke, P. Frenzel. “Clone Detection Using Abstract Syntax Suffix Trees”. In Proceedings of the 13th Working Conference on Reverse Engineering (WCRE), pp. 253-262, 2006.
- J. Krinke. “Identifying Similar Code with Program Dependence Graphs”. In Proceedings of the 8th Working Conference on Reverse Engineering (WCRE), pp. 301-309, 2001.