An empirical study on the characteristics of reusable code clones

Chunli Yu
cunli.yu@queensu.ca

Haoxiang Zhang
haoxianghz@gmail.com

Ying (Jenny) Zou
ying.zou@queensu.ca
Inappropriate code cloning presents both financial and reputation loss for the organization.

- At least 3,156 projects have similar copies of Heartbleed buggy code.

```c
memcpy(bp, pl, payload);
```
<table>
<thead>
<tr>
<th>Original source</th>
<th>Clone Type 1</th>
<th>Clone Type 2</th>
<th>Clone Type 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4: void sumProd(int n) {</td>
<td>void sumProd(int n) {</td>
<td>void sumProd(int n) {</td>
<td>void sumProd(int n) {</td>
</tr>
<tr>
<td>5: float sum = 0.0;</td>
<td>float sum = 0.0; // C1</td>
<td>int s = 0; // C1</td>
<td>int s = 0; // C1</td>
</tr>
<tr>
<td>6: float prod = 1.0;</td>
<td>float prod = 1.0; // C2</td>
<td>int p = 1; // C2</td>
<td>int p = 1; // C2</td>
</tr>
<tr>
<td>7: for (int i = 1; i &lt;= n; i++) {</td>
<td>for (int i = 1; i &lt;= n; i++) {</td>
<td>for (int i = 1; i &lt;= n; i++) {</td>
<td>for (int i = 1; i &lt;= n; i++) {</td>
</tr>
<tr>
<td>8: sum = sum + i;</td>
<td>sum = sum + i;</td>
<td>s = s + i;</td>
<td>s = s + i * i;</td>
</tr>
<tr>
<td>9: prod = prod * i;</td>
<td>prod = prod * i;</td>
<td>p = p * i;</td>
<td>foo(s, p);</td>
</tr>
<tr>
<td>10: foo(sum, prod);</td>
<td>foo(sum, prod);</td>
<td>foo(s, p);</td>
<td>foo(s, p);</td>
</tr>
<tr>
<td>11: }</td>
<td>}</td>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>12: }</td>
<td>}</td>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

- Code clones are created by copy-and-paste activities.

Often 20% - 30% redundancy
Code cloning makes software maintenance difficult

Developers face a large amount of code clones to manage.

- Most developers are contributing voluntarily
- Focus is towards fixing the issues
- Clone detection tools produce a large amount of clones
- It is not possible to check all clones
Limitation of existing approaches to improve clone quality

- Several prior studies attempt to identify problematic clones for refactoring.
- The prior studies provide reuse suggestions mainly based on clone prevalence and/or API-usage related to clones.
- However, reusing code clones reliably from the quality perspective remains unstudied.
To assist development and maintenance team in enhancing code quality, we aim to **pinpoint high-quality code clones** for reliable reuse.
Research Challenges

➢ Difficult to manually examine clones to create a labeled dataset.

➢ The manually labelled results could be subjective and unreliable.
Applying classification models to identify reusable code clones

➢ Automatically classify the clones for more reliable reuse considering both functional requirements and non-functional requirements

➢ Prioritize reusable clones for immediate attentions in quality improvement
Criteria for selecting reusable clones

- Prevalence of code clones
- Lifecycle of code clones
- Fault Resilience
Clones that survive for a longer time period imply higher reusability.
Clone prevalence

- Refers to the **frequency** of reusing code fragments.
- Assesses **functionality** usefulness.
- Calculated by the **number** of clone siblings.
Clone fault-resilience

• Reusing bug-prone clones can harm the quality of a project.

• Frequent buggy changes present signs of inferior code duplicates that are to be sifted out.

\[
\text{Fault-resilience} = \frac{\# \text{ Non-buggy commits}}{\# \text{ Buggy commits}}
\]
Data collection approach

Source code repository (Git) → Extract commit history → Detect clones → Clone groups → Identify clone genealogies → Extract metrics → Features

Analyze reusable functionality (RQ3) → Understand the characteristics of reusable code clones (RQ2) → Classified clones → Classify clones to identify reusable code clones (RQ1) → Labeled dataset → Create labeled dataset
Criteria for selecting subject systems

- Github Java projects
- Commits > 1,000
- Issues > 1,000
- Pull requests > 1,000
- Source lines of code (SLOC) > 100,000

In total, we have 27 subject systems
Detecting code clones

NiCad clone detector is leveraged to detect Type I, Type II, Type III code clones on method level.
Calculating clone metrics

Clone Metrics
- # clone instances
- # of followers
- # length of common paths

Product Metrics
- LOC
- Complexity
- # Fanin
- # Fanout
- ...

Process Metrics
- # contributors
- ...

16
Constructing training data set

Lifecycle
- Longevious
- Meteoric

Prevalence
- Prosperous
- Rare

Fault Resilience
- Metamorphosed
- Vicissitudinous
Tree-based classifiers are leveraged in our case study.
- Decision Tree
- Random Forest
- XGBoost
- CatBoost
- LightGBM
- AdaBoost

10 * 10 cross validation is used to fine-tune our models.
Research questions (RQ)

RQ1

How well can we classify the reusable code clones?
RQ1. How well can we classify the reusable code clones
RQ1. The AUC achieved by ML classifiers in classifying the reusable code clones.
RQ1. The AUC achieved by ML classifiers in classifying the reusable code clones.

Our Random Forest classifier achieves the best AUC (i.e., 0.79) in determining reusable code clones.
RQ2. Features that have the most explanatory power in distinguishing reusable/non-reusable clones

The features number of followers, number of contributors, and length of common paths provide the most explanatory power.
RQ3. The functionalities of reusable code clones

<table>
<thead>
<tr>
<th>Function Category</th>
<th>Category Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>Retrieve related information to the given object</td>
</tr>
<tr>
<td>Config</td>
<td>Prepare and initialize working environment</td>
</tr>
<tr>
<td>Check/Determine</td>
<td>Check specific status</td>
</tr>
<tr>
<td>Add/Remove from collection</td>
<td>Append/add/insert/remove/delete/exclude</td>
</tr>
<tr>
<td>Handle events</td>
<td>Listen to an event and take handling actions</td>
</tr>
<tr>
<td>Convert</td>
<td>Convert an object from one type to another</td>
</tr>
<tr>
<td>Read/Write</td>
<td>Read/write from/to db, file, stream, buffer</td>
</tr>
<tr>
<td>Math</td>
<td>Process calculations</td>
</tr>
<tr>
<td>Encode/Decode</td>
<td>Encode or decode an object</td>
</tr>
<tr>
<td>Visit</td>
<td>Traverse a collection</td>
</tr>
<tr>
<td>Cleanup</td>
<td>Cleanup working environment</td>
</tr>
</tbody>
</table>
11 functionality categories derived from the clone groups reveal the intention of the code, the top three categories are: search, event-handling, and convert.
Reusing code clones to improve code quality

To assist development and maintenance teams in enhancing code quality, we aim to pinpoint high-quality code clones for reliable reuse.

RQ2. Features that have the most explanatory power in distinguishing reusable/non-reusable code clones.

The features number of followers, number of contributors, and length of common paths provide the most explanatory power.

RQ1. The AUC achieved by ML classifiers in classifying the reusable code clones.

Our Random Forest classifier achieves the best AUC (i.e., 0.79) in determining reusable code clones.

11 functionality categories derived from the clone groups reveal the intention of the code, the top three categories are: search, event-handling, and convert.