Assignments

- Assignment 2: Graded!
  - Average 7, standard deviation 1.3
- Assignment 3: March 15, midnight
- Assignment 4 will be published on March 15
  - Code duplication

Where are we now?

- Last week: architecture
  - Behaviour
    - static/dynamic,
    - sequence diagrams/state machines,
    - focusing/visualization
- This week: code duplication
  - Occurs in the code
  - Can reflect suboptimal architecture

Duplication?

- Beck and Fowler, “Stink Parade of Bad Smells”: 1
- Common?

<table>
<thead>
<tr>
<th>Author</th>
<th>System</th>
<th>Min. length (lines)</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ducasse et alii (1999)</td>
<td>Payroll</td>
<td>10</td>
<td>59</td>
</tr>
</tbody>
</table>

- Frequent and problematic!

A rose by any other name

- Popular terms
  - Software redundancy
    - Not every type of redundancy is harmful
  - Code cloning = Code duplication
    - Clone is identical to the original form
- Questions
  1. When are two fragments to be considered as clones?
  2. When is cloning harmful/useful?
  3. How do the clones evolve?
  4. What can one do about clones: ignore, prevent, eliminate?
  5. How to detect and present the clones?
Clones Award

2

Clones?

Clones?

1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596

if ( !GlobalConfig.DEBUG_LEVEL & DEBUG_WARNINGS ) {
    printf("%s: WARNING : SHMEM msg received after sending ANSWER "%s"
",
        acModuleName, sMsgList.asTxMsg[ uiMsgHandle ].name );
    fclose( fp ) ;
    if ( debug_flag ) {
        printf(" result of parser ");
        if ( ! print_tree( FALSE ) ) {
            print_error(stdout , 0) ;
            return FALSE ;
        }
    }
    if ( ! type_check( ) ) {
        print_error(stdout , 0) ;
        return FALSE ;
    }
    if ( debug_flag ) {
        printf(" result of type check"); print(" result of type check") ;
        if ( ! print_tree( TRUE ) ) {
            print_error(stdout , 0) ;
            return FALSE ;
        }
    }
    return FALSE ;

4278 case TYPE_SHMEM:
4279 if (GlobalConfig.DEBUG_LEVEL & DEBUG_WARNINGS) {
4280     printf("%s: WARNING : SHMEM msg received after sending ANSWER "%s"
", 
        acModuleName, sMsgList.asTxMsg[ uiMsgHandle ].name );
4281     fclose( fp ) ;
4282     if ( debug_flag ) {
4283         printf(" result of parser ");
4284         if ( ! print_tree( FALSE ) ) {
4285             print_error(stdout , 0) ;
4286             return FALSE ;
4287         }
4288     }
4289     return( MIRPA_OK ) ;
4290 }

4270 case TYPE_MSGOK:
4271 if (GlobalConfig.DEBUG_LEVEL & DEBUG_WARNINGS) {
4272     printf("%s: INFO : MSG_OK received after sending ANSWER "%s"
", 
        acModuleName, sMsgList.asTxMsg[ uiMsgHandle ].name );
4273     return( MIRPA_OK ) ;
4274 }

if ( !parse( ) ) {
    print_error(stdout , 0) ;
    return FALSE ;
}

if ( !type_check( ) ) {
    print_error(stdout , 0) ;
    return FALSE ;
}

if ( debug_flag ) {
    printf(" result of parser ");
    if ( ! print_tree( FALSE ) ) {
        print_error(stdout , 0) ;
        return FALSE ;
    }
}

Clones?

By Bob Jenkins, 1996. hashtab.h Public Domain

struct hashtable

create_hashable(unsigned int minsize, 
    unsigned int "(hashfunction) (void)",  
    int (*key_eq_fn) (void*)void);  

void hashtable_destroy(struct hashtable *h, int free_values);

/* Copyright (C) 2002 Christopher Clark <firstname.lastname@cl.cam.ac.uk> */ 

/* Copyright (C) 2002 Christopher Clark <firstname.lastname@cl.cam.ac.uk> */ 

Method clones [Balazinska et al. 1999]

3-9 – one token only

10-12 – aggregated changes
• Interface: 3-6
• Implementation: 7-9
• Interface and implem.: mix

Types are too rough!

If we want to eliminate the duplicates we need to understand the differences between them!

Type 1

Type 2

Type 3

Type 4
Structural classification [Kapser et alii 2003]

- Alternative based on the locations of the clones.
- Intra-file or inter-file cloning
- Type of location:
  - function, declaration, macro, hybrid, other (typedef)
- Type of the code sequence
  - initialization, finalization, loop, switch

Q1: Two fragments are clones if...

- Type 1: They are identical up to whitespace/comments
- Type 2: They are structurally identical (rename variables, types or method calls)
- Type 3: They are similar but statements/expressions could have been added, removed or modified
- Type 4: They implement the same concepts

- Alternative classifications have been proposed:
  - [Balazinska et al. 1999] based on the differences
  - [Kapser et al. 2003] based on the location

Q2: Is cloning bad? Good reasons for cloning

- Improves reliability
  - n-version programming, IEC 61508
- Reduces development time
  - “Copy and modify” is faster than “generalize”
- Avoids breaking the existing code
  - Re-testing effort might be prohibitive
- Clarifies structure
  - E.g., disentangles dependencies (but do not overdo!)
  - By lack of choice
    - Programming language does not provide appropriate flexibility mechanisms

However (bad news)...

- More code
  - More effort required to comprehend, test and modify
  - Higher resource usage
- Interrelated code
  - Bug duplication
  - Incomplete or inconsistent updates
- Indicative of
  - Poor or decaying architecture
  - Lack of appropriate knowledge sharing between the developers

Even more: duplication and bugs

- [Monden et al. 2002]
  - 2000 modules, 1MLOC Cobol
  - Most errors in modules with ≥ 200 LOC cloned
  - Many errors in modules with ≤ 50 LOC cloned
  - Least errors in modules with 50-100 LOC clones
  - No explanation of this phenomenon
- [Chou et al. 2001]
  - Linux and Open BSD kernels
  - In presence of clones: one error ⇒ many errors

Q3. How do the clones evolve?

Li et al. 2006

Li et al. 2006

a) Linux
b) Linux “drivers”
c) Free BSD
d) Free BSD “sys”

Increase followed by stabilization
Q4. What can we do about clones?

• Ignore: the simplest way
• Correct (eliminate):
  • Manual: design patterns
  • Automated:
    – Type 1 or 2 (variable names): function abstraction
    – Type 2 (types) or 3: macros, conditional compilation
    – The programming language should support it
    – Can make the code more complex
    – Develop code generators
    – Challenges:
      – how to invent meaningful names?
      – how to determine the appropriate level of abstraction?

• Prevent:
  • Check on-the-fly while the code is being edited
  • Check during the check-in

• Manage
  • Link the clones (automatically or manually)
  • Once one of the clones is being modified the user is notified that other clones might require modification as well.

Questions and answers so far…

1. When are two fragments to be considered as clones?
   • Type 1, 2, 3, 4
   • More refined classification possible

2. When is cloning harmful/useful?
   • reliability, reduced time, structure?, code preservation
   • more interrelated code, more bugs

3. How do the clones evolve?
   • Increase followed by stabilization

4. What can one do about clones?
   • Ignore, eliminate, prevent (check on the fly), manage (link and notify the user upon change)

Basic challenges in clone detection

• Pairwise comparison of classes, functions, lines
  • Naïve way: O(n^2)
  • Might become prohibitive for large systems

• Type 2: How to abstract from var. names, types, …?
  • Rename all variables to XXX?
  • We still want to know whether the same variable appeared in different statements or not?

• Type 3: Clones can be combined into larger clones
  • Clones can have “gaps”
  • Identity vs. Similarity – similarity measures?

Clone detection techniques

• 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
  • [Baxer 1996]
• AST+Tokens combined [Koschke et al. 2006]
• Program Dependence Graph
  • [Krinke 2001]
Programs are just text!
“Programming language independent”

[Ducasse et al, 1999]
• Remove whitespaces and comments
• Calculate hashes for code lines
• Partition lines into classes based on hashes
• Compare lines in the same partition
• Visualize using dot plot

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the house that Jack built.
This is the rat
That ate the malt
That lay in the house that Jack built.
This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.
Dot plot patterns

<table>
<thead>
<tr>
<th>Ducasse et al., 1999</th>
</tr>
</thead>
<tbody>
<tr>
<td>[6x6]</td>
</tr>
<tr>
<td>[6x6]</td>
</tr>
</tbody>
</table>

- Identical code clones, Type 1
- Modified clones Type 2-3
- Recurrent code (break; preprocess or)
- Code has been inserted, or deleted Type 3

Advantages and disadvantages

- **Good news**
  - Language independent
  - Can detect Type 1, 2, 3 clones

- **Bad news**
  - Granularity: line of code, cannot detect duplication between parts of lines
  - Almost no distinction between “important” and “not important” code parts
    - Variable names
    - Syntactic sugar: if (a==0) {b

Alternative textual comparison approach

- [Marcus and Maletic 2001]: Clones discuss the same concepts
- Higher-level clones: Type 4!
- Identifier names should be the same!
  - If/while/... can be neglected
- Latent semantic analysis (Information retrieval)
- Mosaic 2.7, C, 269 files

Extending the text-based approach

- Program structure instead of text
- Metrics instead of hash-functions [Mayrand et al. 1996]
  - Name: identical or not
  - Layout (5 metrics):
    - avg variable name length, num of blank lines...
  - Expression (5 metrics):
    - num of calls, num of executable statements, ...
  - Control flow (11 metrics):
    - num of loops, num of decisions, ...
  - Many metrics ⇒ lower chance of occasional collisions

Metrics-based clone detection

- **Problems:**
  - Metrics are not independent (num uni calls ≤ num calls)
  - “Allowed range” is arbitrarily chosen
  - Precision?
    - Code₁ = Code₂ ⇒ Metrics(C₁) = Metrics(C₂)
    - Code₁ ≠ Code₂ ⇒ Metrics(C₁) ≠ Metrics(C₂)
    - Metrics(C₁) = Metrics(C₂) ⇒ Code₁ = Code₂?
    - Metrics(C₁) ≠ Metrics(C₂) ⇒ Code₁ ≠ Code₂??
  - Precision can be improved if metrics are combined with textual comparison
    - Still O(n²)
  - But n is small for the “good choice” of metrics
More fine-grained approaches: Tokens!

- [Baker 1995]
- We want to recognize $x=x+y$ and $u=u+v$ as clones

- Identify tokens in the code
- Ignore the keywords
- Split structure and parameters

\[
\begin{align*}
\text{split}\left(\text{list}\right) & \\
\text{if} \left(\text{\textless} \beta\right) & \left\{ x = x + y \right\}
\end{align*}
\]

More fine-grained approaches: Tokens!

- [Baker 1995]
- We want to recognize $x=x+y$ and $u=u+v$ as clones

- Identify tokens in the code
- Ignore the keywords
- Split structure and parameters
- For every structure invent an identifier

\[
\begin{align*}
\text{\textit{j length list}} & \\
\text{j3 x y}
\end{align*}
\]

More fine-grained approaches: Tokens!

- [Baker 1995]
- We want to recognize $x=x+y$ and $u=u+v$ as clones

- Identify tokens in the code
- Ignore the keywords
- Split structure and parameters
- For every structure invent an identifier

\[
\begin{align*}
\text{\textit{j length list}} & \\
\text{j3 x y}
\end{align*}
\]

More fine-grained approaches: Tokens!

- [Baker 1995]
- We want to recognize $x=x+y$ and $u=u+v$ as clones

- Identify tokens in the code
- Ignore the keywords
- Split structure and parameters
- For every structure invent an identifier

\[
\begin{align*}
\text{\textit{j length list}} & \\
\text{j3 x y}
\end{align*}
\]

More fine-grained approaches: Tokens!

- [Baker 1995]
- We want to recognize $x=x+y$ and $u=u+v$ as clones

- Identify tokens in the code
- Ignore the keywords
- Split structure and parameters
- For every structure invent an identifier

\[
\begin{align*}
\text{\textit{j length list}} & \\
\text{j3 x y}
\end{align*}
\]

More fine-grained approaches: Tokens!

- [Baker 1995]
- We want to recognize $x=x+y$ and $u=u+v$ as clones

- Identify tokens in the code
- Ignore the keywords
- Split structure and parameters
- For every structure invent an identifier

\[
\begin{align*}
\text{\textit{j length list}} & \\
\text{j3 x y}
\end{align*}
\]

More fine-grained approaches: Tokens!

- [Baker 1995]
- We want to recognize $x=x+y$ and $u=u+v$ as clones

- Identify tokens in the code
- Ignore the keywords
- Split structure and parameters
- For every structure invent an identifier

\[
\begin{align*}
\text{\textit{j length list}} & \\
\text{j3 x y}
\end{align*}
\]

More fine-grained approaches: Tokens!

- [Baker 1995]
- We want to recognize $x=x+y$ and $u=u+v$ as clones

- Identify tokens in the code
- Ignore the keywords
- Split structure and parameters
- For every structure invent an identifier

\[
\begin{align*}
\text{\textit{j length list}} & \\
\text{j3 x y}
\end{align*}
\]

More fine-grained approaches: Tokens!

- [Baker 1995]
- We want to recognize $x=x+y$ and $u=u+v$ as clones

- Identify tokens in the code
- Ignore the keywords
- Split structure and parameters
- For every structure invent an identifier

\[
\begin{align*}
\text{\textit{j length list}} & \\
\text{j3 x y}
\end{align*}
\]

More fine-grained approaches: Tokens!

- [Baker 1995]
- We want to recognize $x=x+y$ and $u=u+v$ as clones

- Identify tokens in the code
- Ignore the keywords
- Split structure and parameters
- For every structure invent an identifier

\[
\begin{align*}
\text{\textit{j length list}} & \\
\text{j3 x y}
\end{align*}
\]
So far only Type 1 and Type 2 clones

- Type 3 clones – combination of Type 1/2 clones

\[ d_1 = d_2 \]

\[ \max(d_1, d_2) \leq \text{threshold} \]

Type 3 clones can be recognized if
- \( d_1 = d_2 \)
- \( \max(d_1, d_2) \leq \text{threshold} \)

Baker’s approach

- Very fast:
  - 1.1 MLOC
  - minimal clone size: 30 LOC
  - 7 minutes on SGI IRIX 4.1, 40MHz, 256 MB
- Close to language independence
- Depends solely on the tokenizer
- Can be improved by code normalization
- See next slide
- Can identify duplication across function borders
- Might require pre/post-processing

Code normalization (Kamiya et al. 2002)

- Many ways to express the same intention

\[
\begin{align*}
\text{x = y + x} & \quad \text{x = x + y} \\
(\text{if } (a == 1) \text{ x=1;}) & \quad (\text{if } (a == 1) \{ \text{x=1;} \}) \\
\text{static global variables in C} & \quad \text{Drop “static”}
\end{align*}
\]

Case study: Expert system of an insurance company [Kamiya – CCFinder/Gemini]

- Diacritics elimination
- Product line like variants

Clone detection techniques

- 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
  - [Baxer 1996]
  - AST+Tokens combined [Koschke et al. 2006]
  - Program Dependence Graph
  - [Krinke 2001]

Conclusions

- 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
  - To be continued next week...

Conclusions

- 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
  - To be continued next week...