Lecture 25

Clone Detection

CCFinder
Today’s Agenda (1)

- Recap of Polymetric Views
- Class Presentation
  - Suchitra (advocate)
  - Reza (skeptic)
Today’s Agenda (2)

- CCFinder, Kamiya et al. TSE 2002
Recap of Polymetric Views

- Polymetric view is a customizable software visualization tool enriched with software metrics.
- This tool targets initial understanding of a legacy system.
- This tool can help programmers develop a high-level mental model.
- It is simple, powerful, scalable, and customizable; however, it requires some training to parse these generated views.
Class Presentation

- Suchitra (advocate)
- Reza (skeptic)
CCFinder

- CCFinder: A multilingual token-based code clone detection system for large scale source code, Kamiya et al. TSE 2002
Definition of Code Clones

- There is no precise or consistent definition on what clones are.
- “a code portion in source files that is identical or similar to another code”
- Clone are often operationally defined by a definition of a clone detector.
When and Why do programmers create clones?
When and Why do programmers create clones?

- What we have is slight different what we want.
- When reusing code as a mental macro template
- Due to programming language limitations
- Legacy code is well-tested and often reliable.
- Management reasons
  - A team does not want to create a dependency on another team’s code.
  - A team does not support other teams’ usage scenarios and customization
- Automatic code generation
Why is code cloning a problem during software evolution?
Why is code cloning a problem during software evolution?

- When a fault is found in one system, it may have to be propagated to other counterpart systems.
- When cloned systems require similar changes, all systems need to be modified consistently.
- If you miss to update these clones consistently, missed updates could lead to a potential bug.
- Redundant development efforts
- Code plagiarism
Research problem addressed by CCFinder

- How can we find clones written in popular programming languages in a fast & scalable way?
- Industrial strength
- Million-line size system within affordable computation time and memory
- Can use heuristics for finding helpful clones
- Robust to renaming & small edits
- Limited uses of language-dependent clone detection
Approach

• Language-**dependent** parts
  • Lexical analysis
  • Rule-based source transformation

• Language-**independent** parts:
  • Suffix-tree matching algorithm for matching token sequences
Rule-based Transformations

- Remove package names
- Supplement callees
- Remove initialization lists
- Separate class definitions
- Remove accessibility keywords
- Convert to compound block
Parameter Replacement

```cpp
void print_lines ( const set & s ) {
    int c = 0;
    Const_iterator I = s . begin ( ) ;
    for ( ; i != s . end ( ) ; ++ i ){
        cout << c << "", "
        << * I << endl ;
        ++ c ;
    }
}

void print_table ( const map & m ) {
    int c = 0;
    Const_iterator I = m . begin ( ) ;
    for ( ; i != m . end ( ) ; ++ i ){
        cout << c << "", "
        << i -> first << " "
        << i -> second << endl ;
        ++ c ;
    }
}
```

```cpp
$p$ $p ( p$ $p$ & $p$ ) {
    $p$ $p$ = $p$ ;
    $p$ $p$
    $p$ $p$ = $p$ ( ) ;
    for ( ; $p$ != $p$ . $p$ ( ) ; ++ $p$ ){
        $p$ << $p$ << $p$
        $p$ $p$ << $p$
        $p$ $p$ << $p$
        $p$ $p$
        $p$ $p$
        $p$ $p$
        $p$
    }
    $p$ $p$
    $p$ $p$
}
```
Other minor contributions

- Similar to duploc’s scatter-plot visualization
- Suggestions of metrics for clones
Evaluation (1)

• Research questions

• RQ1: Is CCFinder scalable and can be applied to industry size programs?
  • e.g. Two versions of OpenOffice. 10 million lines in total. 68 minutes
  • e.g. FreeBSD, NetBSD, and OpenBSD

• RQ2: What is the impact of each transformation rule?
Evaluation (2)

- RQ3: Can CCFinder be used for investigating where and how similar code fragments are used among similar software systems such as FreeBSD, NetBSD, and Linux?

- A hypothesis: FreeBSD and NetBSD are more similar to each other than Linux.

- Results: about 40% of source files in FreeBSD have clones with NetBSD; whereas less than 5% of source files in FreeBSD or NetBSD have clones with Linux.
Other Existing Clone Detection Techniques (1)

- String
  - Baker's Dup: a lexer and a line-based string matching tool: it removes white spaces and comments; replaces identifiers; concatenates all files; hashes each line for comparison; and extracts a set of pairs of longest matches using a suffix tree algorithm

- Token
  - CCFinder transforms tokens using a language specific rules and performs a token-by-token comparison
Other Existing Clone Detection Techniques (2)

- **AST**
  - Baxter et al.'s CloneDr parses source code to build an abstract syntax tree, compares its subtrees by characterization metrics.
  - Jiang et al. and Koschke et al.

- **PDG**
  - Komondoor and Horwitz clone detector finds isomorphic PDG subgraphs using program slicing
  - Krinke uses a k-length patch matching to find similar PDG subgraphs.
  - PDG-based clone detectors are robust to reordered statements, code insertion and deletion, intertwined code, non-contiguous code.
Other Existing Clone Detection Techniques (3)

- Metric-based

- Metric-based clone detectors compare various metrics called fingerprinting functions. They find clones at a particular syntactic granularity such as a class, a function, or a method because these fingerprints are often defined for a particular syntactic unit.
My general thoughts on CCFinder

- CCFinder is a robust and scalable clone detector.
- As there is no consistent definition of code clones, finding X% of clones in one system does not mean very much; however,
  - Its case studies show that CCFinder can be applied to industrial size programs.
  - Its case studies show that CCFinder can be used for checking hypotheses about the origin of a system.
Revisiting this course’s goal (1)

• I hope you had a fun learning about state-of-the-art methods and tools in software evolution research.

• You have learned how to break down challenges in constructing and evolving software.

• You have learned how to cope with software engineering problems systematically.

• Now you probably know that building and evolving large scale software systems is challenging, yet there are systematic solutions (tool support and techniques) out there.
Revisiting this course’s goal (2)

- I hope you gained confidence in doing research. Why? I believe that research skills are important for both practitioners and researchers.

- I hope you gained perspectives in identifying and formulating research questions.

- I hope you now have learned how to identify open problems through a literature survey.

- I hope you are more comfortable about reading research papers critically and evaluating research works.

- I hope you learned the importance of evaluation component and how to evaluate research solutions.
Preview for Next Lecture

- We will continue with code duplication research.
- Empirical studies of code clone genealogies, Kim et al. FSE 2005
Announcement

- The peer review form is available on the blackboard.
- Please take your graded homework -- practical uses of software evolution research, part 1.
- Your grade review period ends on Apr 27th 11:50 PM.