ICECCS 2017

Exploring Similar Code

- From Code Clone Detection to Provenance Identification -

Katsuro Inoue
Osaka University
My Background

Ph.D. for Implementation & Optimization of Functional Programs

Software Process Modeling & Formalization

Program Slicing

Mining Soft. Repositories, Code Clone, Code Search

Code Provenance, License Analysis

84  '90  '95  '00  '05  '10  '15  '17
Welcome to ICECCS 2017

The 22nd International Conference on Engineering of Complex Computer Systems (ICECCS 2017) will be held on November 6–8, 2017, Fukuoka, Japan. Complex computer systems are common in many sectors, such as manufacturing, communications, defense, transportation, aerospace, hazardous environments, energy, and health care. These systems are frequently distributed over heterogeneous networks, and are driven by many diverse requirements on performance, real-time behavior, fault tolerance, security, adaptability, development time and cost, long life concerns, and other areas. Such requirements frequently conflict, and their satisfaction therefore requires managing the trade-off among them during system development and throughout the entire system life. The goal of this conference is to bring together industrial, academic, and government experts, from a variety of user domains and software disciplines, to determine how the disciplines' problems and solution ...
Welcome to ICECCS 2017
The 22nd International Conference on Engineering of Complex Computer Systems (ICECCS 2017) will be held on November 6-8, 2017, Fukuoka, Japan. Complex computer systems are common in many sectors, such as manufacturing, communications, defense, transportation, aerospace, hazardous environments, energy, and health care. These systems are frequently distributed over heterogeneous networks, and are driven by many diverse requirements on performance, real-time behavior, fault tolerance, security, adaptability, development time and cost, long life concerns, and other areas. Such requirements frequently conflict, and their satisfaction therefore requires managing the trade-off among them during system development and throughout the entire system life. The goal of this conference is to bring together industrial, academic, and government experts, from a variety of user domains and software disciplines, to determine how the disciplines' problems and solution...
Polystyrene-supported GaCl₃ as a highly efficient and recyclable heterogeneous Lewis acid catalyst for one-pot synthesis of N-substituted pyrroles

Ali Rahmatpour

Polymer Science and Technology Division, Research Institute of Petroleum Industry (RIPPI), 14065-6137 Tehran, Iran

ABSTRACT

A new and environmentally friendly method for the preparation of N-substituted pyrroles (Py)s from the one-pot condensation reactions of N,N-dialkylamino aldehydes with amine and diamine in the presence of polystyrene-supported gallium trichloride (PS/GaCl₃) as a highly active and recyclable heterogeneous Lewis acid catalyst is presented. The new protocol has the advantages of easy availability, stability, reusability and ease of catalyst isolation, high selectivity, high yields, and simple experimental and work-up procedure.

1. Introduction

Pyrroles are an important class of nitrogen-containing heterocyclic compounds. They constitute the unit of many natural products, synthetic polymers, and drugs. and serve as building blocks for porphyrin synthesis [1,2]. Pyrroles of this family have wide applications in medicinal chemistry, being used as antimicrobial, antifungal, anticyclization agents, and as antibacterial [3]. The pyrrole ring consists of two nitrogen atoms and two carbon atoms, forming a five-membered ring with partial double bonds.

These compounds can be prepared from the Hantzsch reaction [4], L1-dipolar cycloaddition reactions [7],aza-Wittig reactions [8], annulations reactions [9], and other multistep operations [10]. Despite these developments, the Paal-Knorr reaction remains one of the most significant and simple methods for the synthesis of pyrroles. Polystyrene-supported catalysts have been reported in recent years, as the preparation of catalysts [29] and immobilization of catalysts on solid supports. However, most of these catalysts have been immobilized on polymer supports such as polyethylene, polystyrene, and other polymers. However, all these catalysts have not been reported to be used as an efficient and high-yielding catalyst.
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>

int main() {
    int tot_chars = 0;    /* total characters */
    int tot_lines = 0;    /* total lines */
    int tot_words = 0;    /* total words */
    int boolean;
    /* EOF == end of file */
    int n;
    while ((n = getchar()) != EOF) {
        tot_chars++;
        if (isspace(n) && !isspace(getchar())) {
            tot_words++;
        }
        if (n == '¥n') {
            tot_lines++;
        }
        if (n == '-') {
            tot_words--;
        }
    }
    printf("Lines, Words, Characters¥n");
    printf(" %3d %3d %3d¥n", tot_lines, tot_words, tot_chars);
    return 0;
}
Similarity of Code Snippets
Matters?
Yes, code similarity matters!
Our Code Clone Research
**Code Clone**

- A code fragment with an *identical* or *similar* code fragment

- Created
  - Accidentally: small popular idioms
    - e.g., \[\text{if}(\text{fp} = \text{fopen}("file","r") == \text{NULL})\{\]
  - Intentionally: copy & paste practice
Opening
Issue at Company X

• Maintained code clones by hand

Hand-made clone list

20 years

M-LOC system

Check
No tools available in ‘99
Develop Ourselves! CCFinder
Clone Detection Policies

• Determine detection granularity
  – char / **token** / line / block / function / ...
• Cut off small clones (probably by non-intentional idioms)
• Find all possible clone sets at the same time
• Use efficient and scalable algorithm
Classification of Clones

- **Type 1**: Exact copy, only differences in white space and comments
- **Type 2**: Same as type 1, but also variable renaming
- **Type 3**: Same as type 2, but also deleting, adding, or changing few statements
- **Type 4**: Semantically identical, but not necessarily same syntax.
Example of Clones (type 2)
A Clone Detection Process

1. static void foo() throws RESyntaxException {
2.   String a[] = new String[] { "123,400", "abc", "orange 100" };
4.   int sum = 0;
5.   for (int i = 0; i < a.length; ++i)
6.       if (pat.match(a[i]))
7.           sum += Sample.parseNumber(pat.getParen(0));
8.   System.out.println("sum = " + sum);
9. }
10. static void goo(String[] a) throws RESyntaxException {
11.   RE exp = new RE("[0-9]+")
12.   int sum = 0;
13.   for (int i = 0; i < a.length; ++i)
14.       if (exp.match(a[i]))
15.           sum += parseNumber(exp.getParen(0));
16.   System.out.println("sum = " + sum);
17. }

Software Engineering Laboratory, Department of Computer Science, Graduate School of Information Science and Technology, Osaka University
Match Detection Algorithms

Matching Table

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td>%</td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>x</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>y</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>

Suffix Tree

xyxyz% → 1 → x → 2 → xyz% → 3 → y → 4

xyz% → 3 → y → 5 → z% → 6

z% → 6 → z% → 7

%
Initial Result

• Initial prototype had been implemented in a few weeks

• Limited scalability
  - Refined & tuned
  - Bought powerful workstation and expensive memory
Clones between Unix Kernels

FreeBSD

Linux

NetBSD
Let’s Submit to IEEE TSE!


- Initial submission in July, 2000
- Major revision request in Jan., 2001
- Minor revision request in Aug., 2001
- Accepted in Sept., 2001
- Published in July, 2002

1,413 citations (2017, Nov. 3)
22nd highly-cited paper in SE
Extending CCFinder

- Prototype was extended, tuned, and refined as a practical tool
- Target languages
  - Initially C
  - Adapted to C++, COBOL, Java, Lisp, Text
- Added GUI and result visualizer
Promoting Code Clone Technology

• Collaboration with many companies
  - NTT, Fujitsu, Hitachi, Samsung, Microsoft Research, ...

• International Workshop on Software Clones IWSC (‘02~)

• Code clone seminars (9 times, ‘02~’07)
CCFinder & GEMINI

[CCFinder Website] -> Japanese Page
Toshihiro Kamiya

CCFinder

CCFinder is a tool for detecting code clones and has the following features:

Scalability
CCFinder can be applied to huge source codes. From a source code with approximately a million lines, CCFinder can detect code clones within several minutes to several hours using a PC/AT compatible.

Applicability to various programming languages
By lexical analysis and transformation based on the syntax of the programming languages, CCFinder can extract code clones correctly from source files, even in cases where the names of variables have been changed. CCFinder can run for C/C++, Java, COBOL, Fortran, etc.

Designed as a command-line tool
In the early stages of the development of CCFinder, tools such as a converter for gnuplot, i.e. a converter from the output of CCFinder to a format for the input of a gnuplot, were used to display the distribution of detected code clones.

Screenshot

CCFinder is implemented in C++, and currently has binary executable files for Windows98/Me/2000/XP. A paper by tKamiya20021 describes the algorithm of CCFinder, the experiments of using CCFinder to compare source codes of OS, and the explanation and application of clone metrics RAD and DFL.

> 5,000 downloads
Finding Duplicate Code by using Code Clone Detection

Visual Studio 2015 | Other Versions


The latest version of this topic can be found at Visual Studio 2017 Documentation. Code clones are separate fragments of code that are very similar. They are a common phenomenon in an application that has been under development for some time. Clones make it hard to change your application because you have to find and update more than one fragment.

Visual Studio Enterprise can help you find code clones so that you can refactor them.

You can either find the clones of a specific fragment, or find all clones in your solution. In addition to discovering direct copies, the clone analysis tool can find fragments which differ in the names of variables and parameters, and in which some statements have been rearranged.

The code clone analyser searches for duplicate code in Visual C# and Visual Basic projects throughout your Visual Studio solution.
Clone Notifier

Clone Notifierは、バージョン間でコードクローン（ソースコード中の重複コード）の差分情報を検出・可視化するツールです。本システムを利用してすることによって、コピー・アンド・ペーストによって新しく発生したクローンや、一貫した修正漏れの可能性があるクローンの情報を検出・可視化することができます。詳細な使用方法はhttps://osdn.net/projects/clonenotifier/downloads/66557/manual.pdfを見ることを推奨します。

2013年版はすべてCCFinder対応バージョン、Ver.2.0は関数クローン検出ツール対応バージョンです。

ダウンロード

最新リリース
clonenotifier ver 2.0（関数クローン検出ツール対応バージョン）（日付: 2016-10-11）
clonenotifier 20131101（CCFinder対応バージョン）（日付: 2013-11-01）
clonenotifier 20131029（CCFinder対応バージョン）（日付: 2013-10-29）
clonenotifier 20131029（CCFinder対応バージョン）（日付: 2013-10-29）
Experience of Finding Inconsistently-Changed Bugs in Code Clones of Mobile Software

Katsuro Inoue1, Yosiki Higo1, Norihiro Yoshida1, Eunjong Che1, Shinji Kasumoto1, Hyongwan Kim2, Wonjin Park2, and Eunha Lee2
1Osaka University 2Samsung Electronics Co.
Osaka, Japan Samsung, South Korea
{inoue, higo, n-yoshida, ejoeh, kasumoto}@ist.osaka-u.ac.jp
(kyongshwan73.kim, wj23.park, leewenha)@samsung.com

Abstract—When we reuse a code fragment, some of the identifiers in the fragment might be systematically changed to others. Failing these changes would become a potential bug in the copied fragment. We have developed a tool CloneInspector to detect such inconsistent changes in the code clones, and applied it to two mobile software systems. Using this tool, we were effectively able to find latent bugs in those systems.

Keywords-Inconsistent Change, Unchanged Ratio, Bug Candidate

I. INTRODUCTION

Software systems for mobile phone (mobile software) are becoming huge and complex, and debugging and maintaining them are getting difficult and expensive.

A mobile software system needs to adapt its features to various circumstances, and so many code clones are being reused. When we reuse a code fragment, some of the identifiers in the fragment might be systematically changed to others. Failing these changes would become a potential bug in the copied fragment. We have developed a tool CloneInspector, which uses a prototype tool, and have built a tool named CloneInspector. In this paper, we will show an overview of CloneInspector, and present our experience of applying CloneInspector to Samsung’s large mobile software.

II. CLONEINSPECTOR

Figure 1 shows the process of CloneInspector.

1) First, code clones in the input source files are detected by code clone detector CCFinder (2). The positions of code clones are generated.

2) Using the positions, code fragments for the detected code clones are tokenized. At the same time, the occurrences of identifiers are examined.

Applications

- Feature: Communication
  - Language: C
  - Size(LOC): 4,275,952
  - Bug Candidates: 63
  - True Bug: 25

- Feature: Application
  - Language: C
  - Size(LOC): 136,554
  - Bug Candidates: 5
  - True Bug: 1

Snapshot of Clone Inspector

- Inconsistencies Candidate List
- Information of Selected Inconsistencies Candidate
- Source Code View

Overview of Clone Inspector

- A tool to fit to the development environment of mobile software in SAMSUNG
  - Detect **inconsistent changes** in code clones
    - Using CCFinder and two metrics
  - Handle a large size input
Papers Related to Code Clone

Number of Papers:

- 00
- 01-05
- 06-10
- 11-15 (highest)
- 16~
Applying to New Fields
Compiler Construction Class

Number of drawn files: 15
Lines of drawn files: 16,030
Tokens of drawn files: 54,489
Compiler Construction Class

S1  S2  S3  S4  S5

Number of drawn Files: 15
Lines of drawn Files: 16,030
Tokens of drawn Files: 54,488
Law Suit of Lace-making Machines
Clones in near systems
System Evolution Analysis — BSD Unix Clustering —

FreeBSD 2.0
FreeBSD 2.0.5
FreeBSD 2.1
FreeBSD 2.2
FreeBSD 3.0
FreeBSD 4.0
4.4BSD Lite
4.4BSD Lite2
NetBSD 1.0
NetBSD 1.1
NetBSD 1.2
OpenBSD 2.0
OpenBSD 2.1
OpenBSD 2.2
OpenBSD 2.3
OpenBSD 2.4
OpenBSD 2.5
OpenBSD 2.6
OpenBSD 2.7
OpenBSD 2.8
NetBSD 1.3
NetBSD 1.4
NetBSD 1.5

Similarity Measure ← Cover Ratio
Chasing Scalability

- CCFinder input limitation: A few M LOC due to the suffix tree algorithm --- on core
- Code clone detection is embarrassingly parallel problem --- divide and concur

Distributed CCFinder with 80 lab machines
FreeBSD Ports Collection / 136 Versions of Linux Kernel

Searching for clone pairs
Where Does This Code Come from and Where Does It Go?  
- Integrated Code History Tracker for Open Source Systems -

Katsuro Inoue, Yusuke Sasaki, Pei Xia, and Yuki Manabe  
Osaka University  
Osaka, Japan  
{inoue, peixia, y-manabe}@ist.osaka-u.ac.jp

Abstract—When we reuse a code fragment in an open source system, it is very important to know the history of the code, such as the code origin and evolution. In this paper, we propose an integrated approach to code history tracking for open source repositories. This approach takes a query code fragment as its input, and returns the code fragments containing the code clones with the query code. It utilizes publicly available code search engines as external resources. Based on this model, we have designed and implemented a prototype system named Ichi Tracker. Using Ichi Tracker, we have conducted three case studies. These case studies show the ancestors and descendants of the code, and we can recognize their evolution history.

Keywords—Code Search; Software Evolution; Open Source System

I. INTRODUCTION

Open source systems are extremely useful resources for the construction of current software systems. Even software systems in the industry increasingly use open source systems due to their reliability and cost benefits [25].

One of usages of the open source systems is to reuse the source code of the open source systems for other projects. We can easily get the source code files of various projects from the repositories on the Internet, such as SourceForge [38] and Maven Central [26]. Those source code files are

Current software engineering tools do not provide sufficient support to explore code history. To know the code origin, we have to specify project names and/or URLs. Also, to know the code evolution, we have to understand the interrelations of open source projects.

Code search engines such as Google Code Search [10] and Koders [3] are very useful tools to explore open source repositories for the origin and evolution of code. However, current code search engines only allows to get keywords and/or code attributes as their inputs, and they return source code files which contains those keywords and attributes. Selecting appropriate inputs for those search engines is not easy task for general users.

In this paper, we will propose an integrated approach to code history tracking for open source repositories. Also, we will present its prototype system named Ichi Tracker (Integrated Code History Tracker). Ichi Tracker takes a code fragment as its query input, and returns a set of cloned code fragments which can be found by popular source code search engines such as SPARS/R [30], Google Code Search [10], and Koders [3]. Ichi Tracker helps us to understand the backward and forward history of the query code fragment.

Using Ichi Tracker, we have performed various case
Developer’s Concerns

Existing Project

New Project

Reuseable?

Developer

To ease concerns, a support system is needed

• Origin
  - Who?
  - When?
  - License?
  - Copyright?

• Evolution
  - Maintenance?
  - Popularity?
  - Newer version?
  ...

Concerns

Software Engineering Laboratory, Department of Computer Science, Graduate School of Information Science and Technology, Osaka University
Code History Tracking System

OSS Repositories
System Overview

Input Query $Q$

- Code Fragment $q_c$
- Code Attributes (Optional)

Output Results $R$

- Code Clones
- Code Attributes

Integrated Code History Tracker

Ichi Tracker

Code Search Engines

- SPARS/R
- Google Code Search
- Koders

Internet

Open Source Repositories

Search Query $SQ$

Search Results $SR$
Current and Future
OSS Dependency

Clone A

System Y

Clone B

System Z

OSS System X

Library P

Library Q
Do developers update their library dependencies?
An empirical study on the impact of security advisories on library migration

Raula Gaikovina Kula³ · Daniel M. German² · Ali Ouni¹,⁴ · Takashi Ishio³ · Katsuro Inoue¹

Abstract  Third-party library reuse has become common practice in contemporary software development, as it includes several benefits for developers. Library dependencies are constantly evolving, with newly added features and patches that fix bugs in older versions. To take full advantage of third-party reuse, developers should always keep up to date with the latest versions of their library dependencies. In this paper, we investigate the extent of which developers update their library dependencies. Specifically, we conducted an empirical study on library migration that covers over 4,600 GitHub software projects and 2,700 library dependencies. Results show that although many of these systems rely heavily on dependencies, 81.5% of the studied systems still keep their outdated dependencies. In the case of updating a vulnerable dependency, the study reveals that affected developers are not likely to respond to a security advisory. Surveying these developers, we find that 69% of the...
(a) LMP for consecutive releases of the google-guava (NR1) library
Commons-beanUtils (V1) versions

- $\mathcal{L}(V1, v1.9.1)$ Java 5+
- $\mathcal{L}(V1, v1.9.2)$ Java 5+

A Library Migration Plot.
OSS Ecosystem

OSS Repository

Project

Depend

Depend

Contribute

Use

Developer

User

Software Engineering Laboratory, Department of Computer Science, Graduate School of Information Science and Technology, Osaka University
Summary
Finding similar code in software matters
Code clone analysis became very popular SE technology
Analyzing code similarity is key to know OSS provenance and evolution
Exploring OSS universe with code similarity analysis
Thank you