ICECCS 2017

Exploring Similar Code

- From Code Clone Detection to Provenance Identification -

Katsuro Inoue

Osaka University

My Background

Formal → Empirical

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

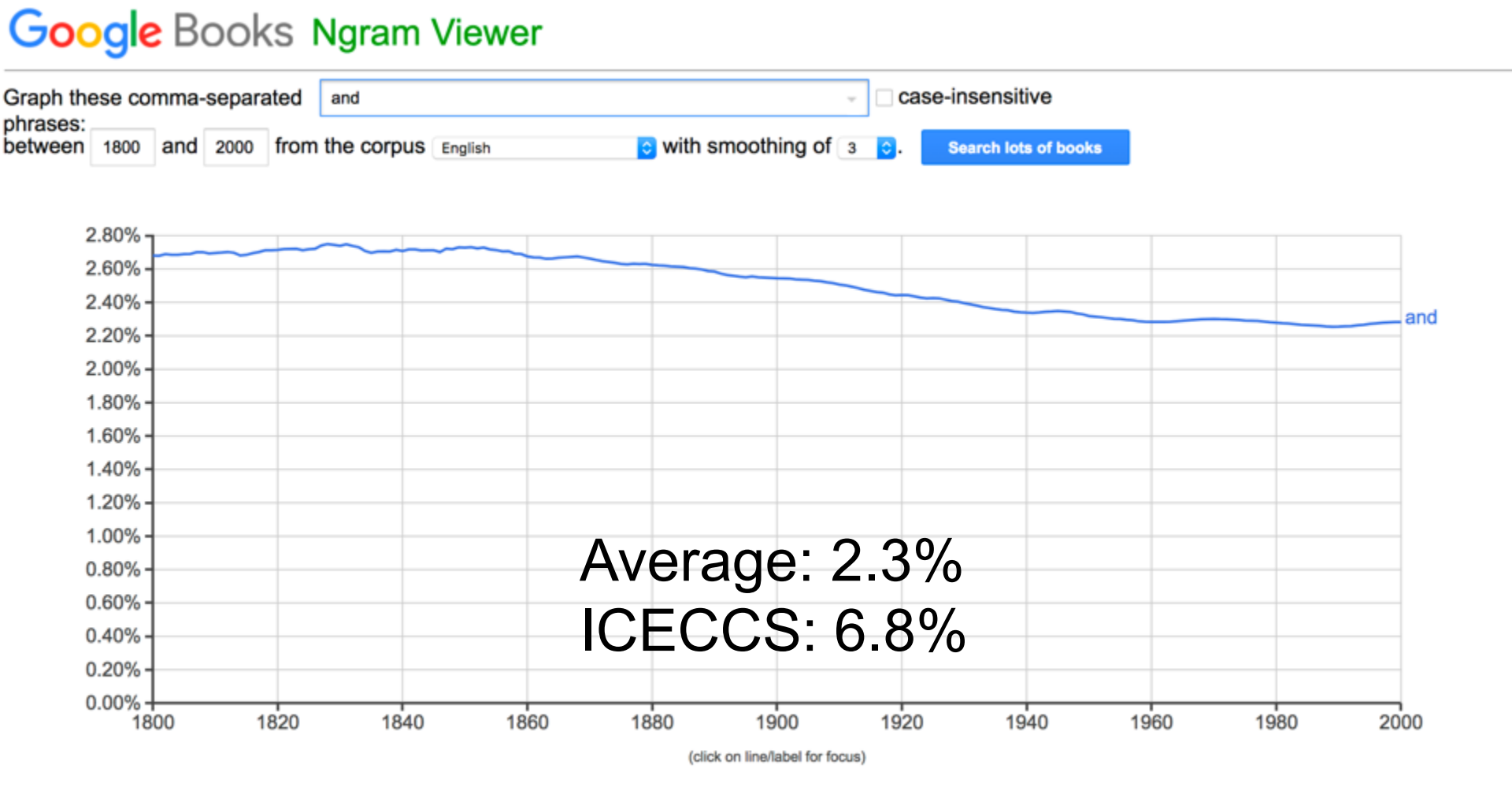
84 '90 '95 '00 '05 '10 '15 '17

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,



iThenticate

iThenticate article

Quotes Excluded
Bibliography Excluded

38% SIMILAR

Polystyrene-supported $GaCl_3$ 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 (RIPI), 14665-1137 Tehran, Iran

ARTICLE INFO

Article history:
Received 27 December 2011
Received in revised form
10 March 2012
Accepted 21 March 2012

Keywords:
Polymer-supported catalyst
Pyrrole
Paal–Knorr condensation reaction
Heterogeneous Lewis acid catalyst

ABSTRACT

A new and environmentally [illegible] method for the preparation of *N*-substituted pyrroles from one-pot condensation reaction of 2,5-hexanedione with amines and diamines in the presence of polystyrene-supported gallium chloride ($PS/GaCl_3$) as a highly active and reusable heterogeneous Lewis acid catalyst is presented. The new protocol has the advantages of easy availability, stability, reusability and eco-friendly of the catalyst, high to excellent yields, simple experimental and work-up procedure.

1. Introduction

Functionalized pyrroles are an important class of nitrogen-containing heterocyclic compounds. They constitute the core unit of many natural products, synthetic materials, and serve as building blocks for porphyrin synthesis [1,2]. Members of this family have wide applications in medicinal chemistry, being used as antimalarial, antiflammatory agents, antibacterial, antifungal, antiviral [3–5]. These compounds can be prepared from the classical Hantzsch procedure [6], 1,3-dipolar cycloaddition reactions [7], aza-Wittig reactions [8], annulations reactions [9], and other multistep operations [10]. Despite these new developments, the Paal–Knorr condensation remains one of the most significant and simple methods which consists the cyclocondensation of primary amines with 1,4-dicarbonyl compounds to produce *N*-substituted pyrroles. Several catalysts have been used to promote this reaction including HCl [11], *p*-TSA [12], H_2SO_4 [13], $Sc(OTf)_3$ [14], $Bi(NO_3)_3{\cdot}5H_2O$ [15], $SnCl_2{\cdot}2H_2O$ [16], $Ti(OPr^i)_4$ [17], $RuCl_3$ [18], $InCl_3$, $InBr_3$, $In(OTf)_3$ [19], zeolite [20], Al_2O_3 [21], montmorillonite K10 [22], silica sulfuric acid [23], layered zirconium phosphate and phosphonate [24], montmorillonite [25], montmorillonite KSF-clay and I_2 [26]. Additionally, the above cyclocondensation process could proceed in ionic liquid [27] or ultrasonic and microwave irradiation [28]. However, despite the potential utility of these catalysts, many of these methodologies for the synthesis of pyrroles are associated with several shortcomings such as low yields, prolonged reaction time, harsh reaction conditions, the requirement of excess of catalysts, the use of toxic and detrimental metal precursors as catalysts, and relatively expensive reagents and high temperature, and tedious work-up leading to the generation of large amounts of toxic metal-containing waste. The main disadvantage of almost all existing methods is that the catalysts are destroyed in the work-up procedure and their recovery and reuse is often impossible, which limit their use under the aspect of environmentally benign processes.

Heterogeneous supported catalysts have been gained much attention in recent years, as they possess a number of advantages in preparative procedures [29,30]. The immobilization of catalysts on solid support improves the available active site, stability, hygroscopic properties, handling, and reusability of catalysts which all factors are important in industry [31]. Therefore, use of supported and reusable catalysts in organic transformations has economical and environmental benefits. A large number of polymer supported Lewis acid catalysts have been prepared by immobilization of the catalysts on a polymer via coordination or covalent bonds [32]. Such polymeric catalysts are usually as active and selective as their homogeneous counterparts while having the distinguishing characteristics of being easily separable from the reaction mixture, recyclability, easier handling, non-toxicity, enhanced stability, and improved selectivity in various organic reactions. Polystyrene is one of the most widely studied heterogeneous and polymeric supports due to its environmental stability and hydrophobic nature

* Tel.: +98 21 44739588; fax: +98 21 44739587.
E-mail address: rahmatpourali@ripi.ir.

0022-328X/$ – see front matter
doi:10.1016/j.jorganchem.2012.03.025

Match Overview

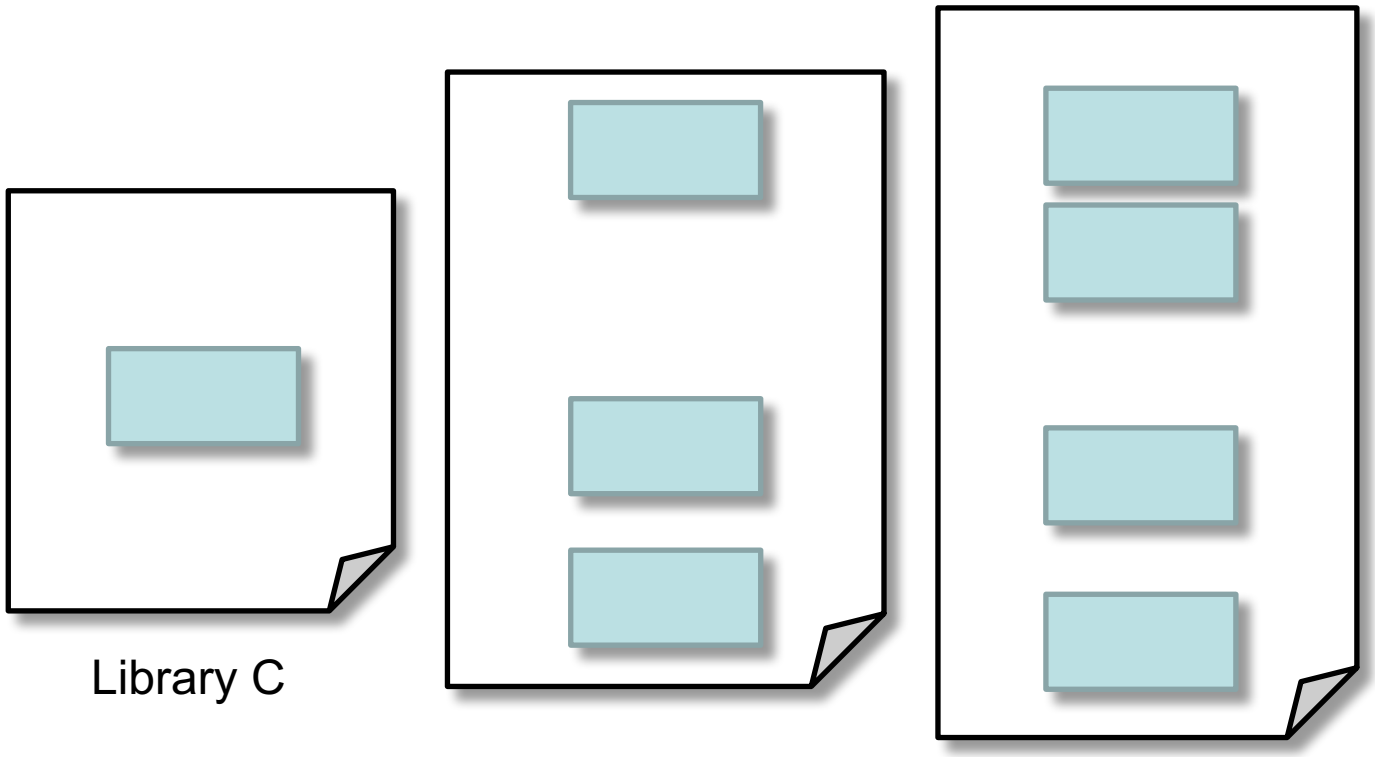
1	**CrossCheck** 135 words Liang Wang. "Polystyrene-supported $AlCl_3$: A highly active a nd reusable heterogeneous catalyst for the one-pot synth	3%
2	**CrossCheck** 131 words Chen, J.. "An approach to the Paal-Knorr pyrroles synthes ... catalyzed by Sc(OTf)"3 under solvent-free conditions", Tetr	3%
3	**CrossCheck** 113 words Borujeni, K.P.. "Synthesis and application of polystyrene s ... pported aluminium triflate as a new polymeric Lewis acid c	2%
4	**CrossCheck** 91 words Liang Wang. "Polymer-supported zinc chloride: a highly acti ve and reusable heterogeneous catalyst for one-pot synth ...	2%
5	**CrossCheck** 76 words Ali Rahmatpour. "An efficient, high yielding, and eco-friendly method for the synthesis of 14-aryl- or 14-alkyl-14H-dibenz	2%
6	**CrossCheck** 73 words Ran Ruicheng. "Polymer-Supported Lewis Acid Catalysts ... . Polystyrene-Gallium Trichloride Complex", Journal of Macr	2%
7	**CrossCheck** 54 words Karimi, B.. "Solid silica-based sulfonic acid as an efficient a nd recoverable interphase catalyst for selective tetrahydr ...	1%
	CrossCheck 53 words	

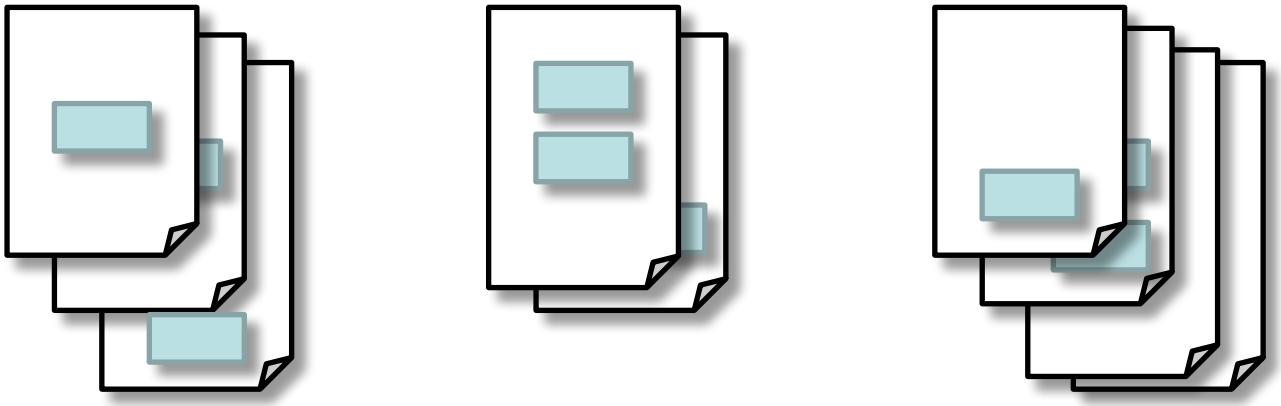
```c
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
int main() {
    int tot_chars = 0;      /* total characters */
    int tot_lines = 0;      /* total lines */
    int tot_words = 0;      /* total words */
    int boolean;
    /* EOF == end of file */
    int n;
    while ((n = getchar()) != 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

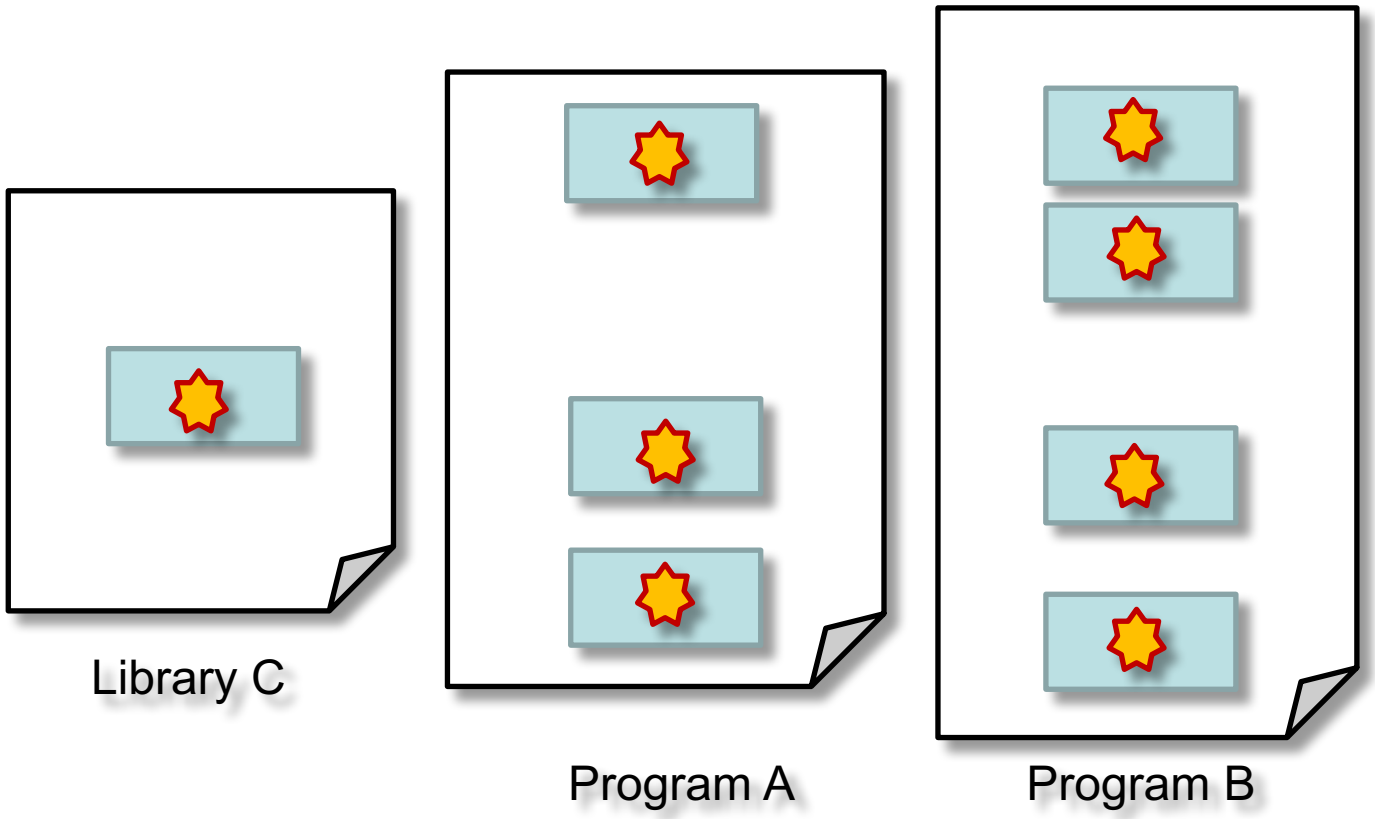
```
public static byte[] mergeLocalFileDataData(ZipExtraField[] data) {
    final boolean lastIsUnparseableHolder = data.length > 0
        && data[data.length - 1] instanceof UnparseableExtraFieldData;
    int regularExtraFieldCount =
        lastIsUnparseableHolder ? da

    int sum = WORD * regularExtraFiel
    for (ZipExtraField element : data
        sum += element.getLocalFileDa
    }

    byte[] result = new byte[sum];
    int start = 0;
    for (int i = 0; i < regularExtraF
        System.arraycopy(data[i].getL
                         0, result,
        System.arraycopy(data[i].getL
                         0, result,
        byte[] local = data[i].getLo
        System.arraycopy(local, 0, re
        start += (local.length + WORD
    }
    if (lastIsUnparseableHolder) {
        byte[] local = data[data.leng
        System.arraycopy(local, 0, re
    }
    return result;
}
```

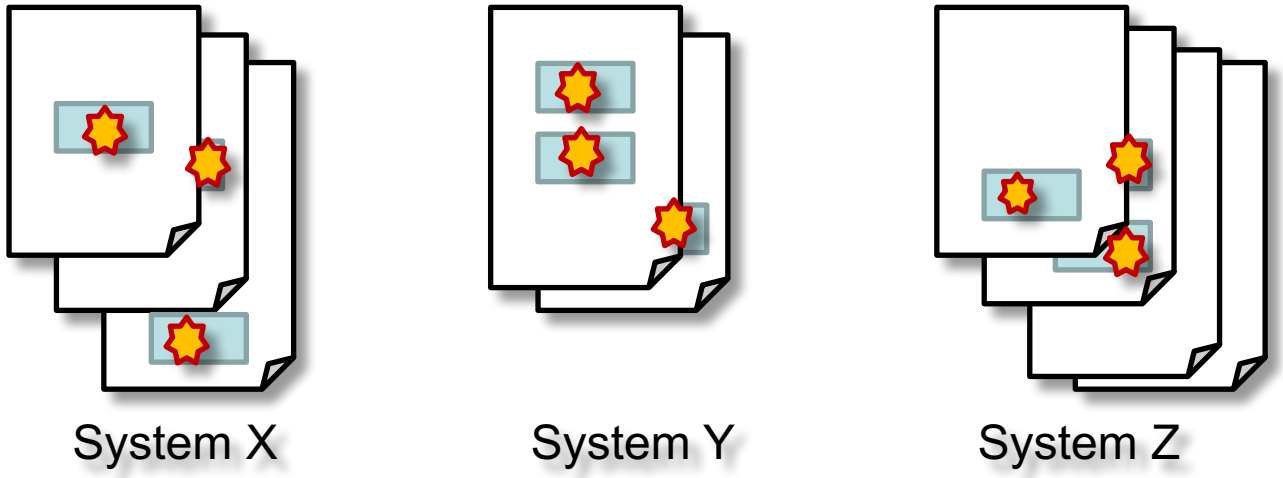
```
public static byte[] mergeCentralDirectoryData(ZipExtraField[] data
    final boolean lastIsUnparseableHolder = data.length > 0
        && data[data.length - 1] instanceof UnparseableExtraFieldDa
    int regularExtraFieldCount =
        lastIsUnparseableHolder ? data.length - 1 : data.length;

    int sum = WORD * regularExtraFieldCount;
    for (ZipExtraField element : data) {
        sum += element.getCentralDirectoryLength().getValue();
    }
    byte[] result = new byte[sum];
    int start = 0;
    for (int i = 0; i < regularExtraFieldCount; i++) {
        System.arraycopy(data[i].getHeaderId().getBytes(),
                         0, result, start, 2);
        System.arraycopy(data[i].getCentralDirectoryLength().getByt
                         0, result, start + 2, 2);
        byte[] local = data[i].getCentralDirectoryData();
        System.arraycopy(local, 0, result, start + WORD, local.leng
        start += (local.length + WORD);
    }
    if (lastIsUnparseableHolder) {
        byte[] local = data[data.length - 1].getCentralDirectoryData
        System.arraycopy(local, 0, result, start, local.length);
    }
    return result;
}
```

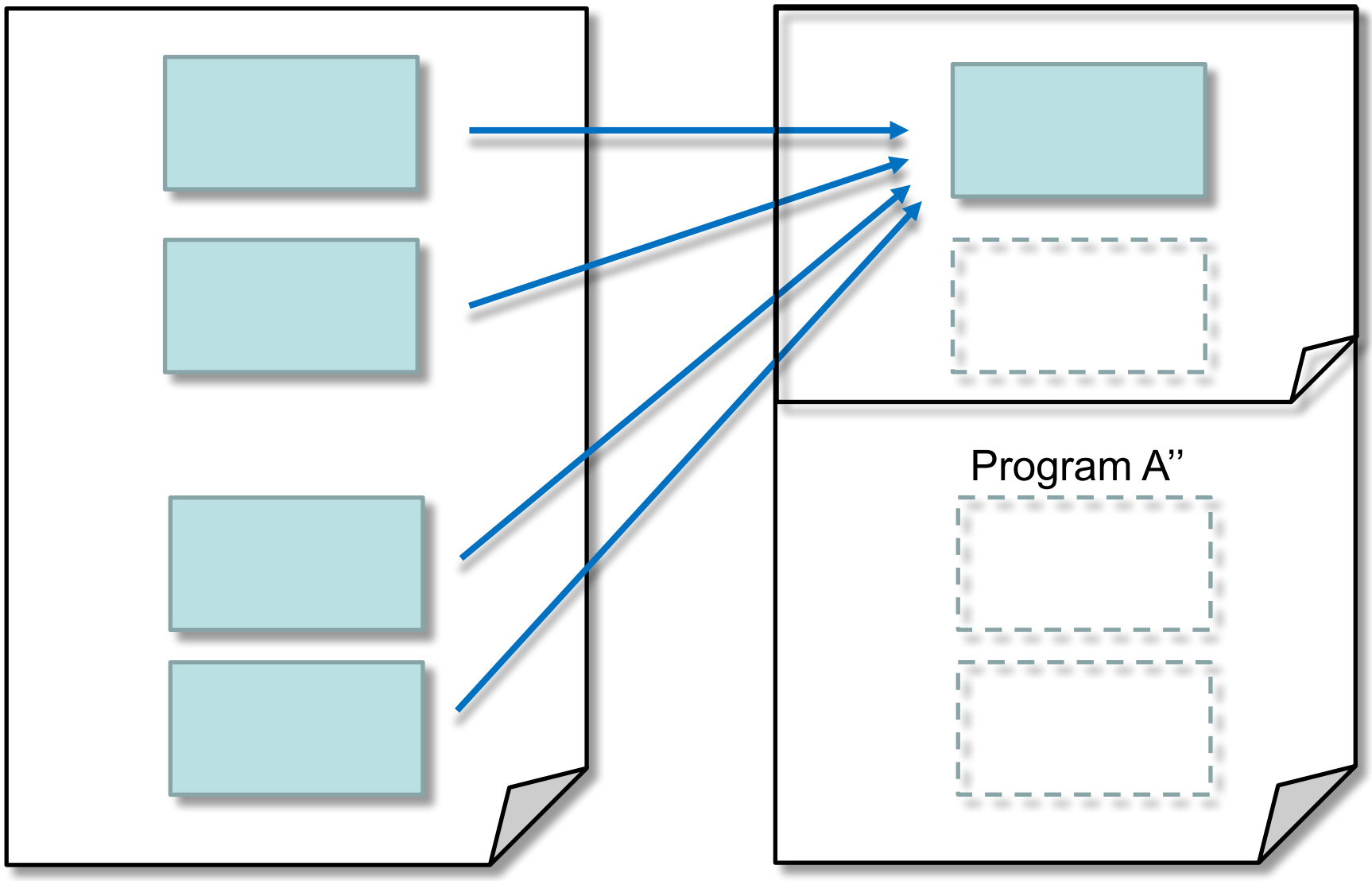

Library C
Program A
Program B


System X
System Y
System Z

Matters?


Library C
Program A
Program B


System X
System Y
System Z


Program A''
Program A
Program A'

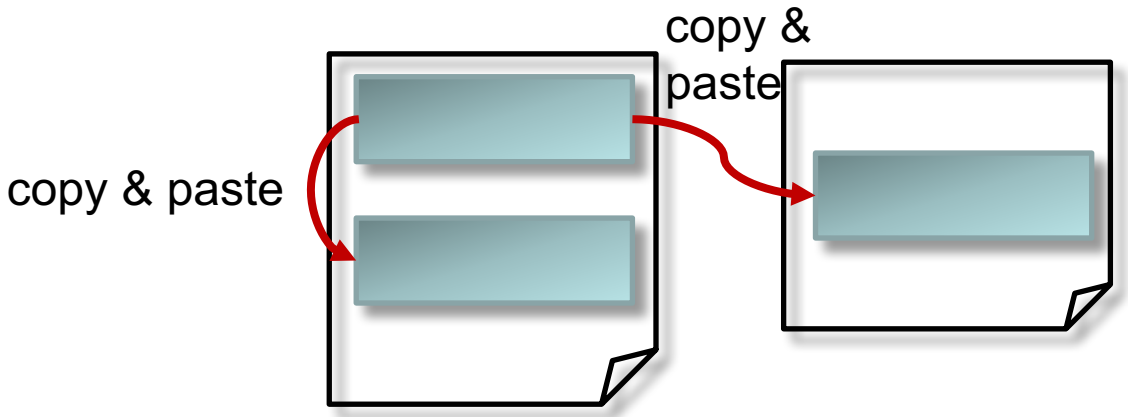
Yes, code similarity matters!

Our Code Clone Research

Code Clone

- A code fragment with an *identical* or *similar* code fragment
- Created
  - Accidentally: small popular idioms

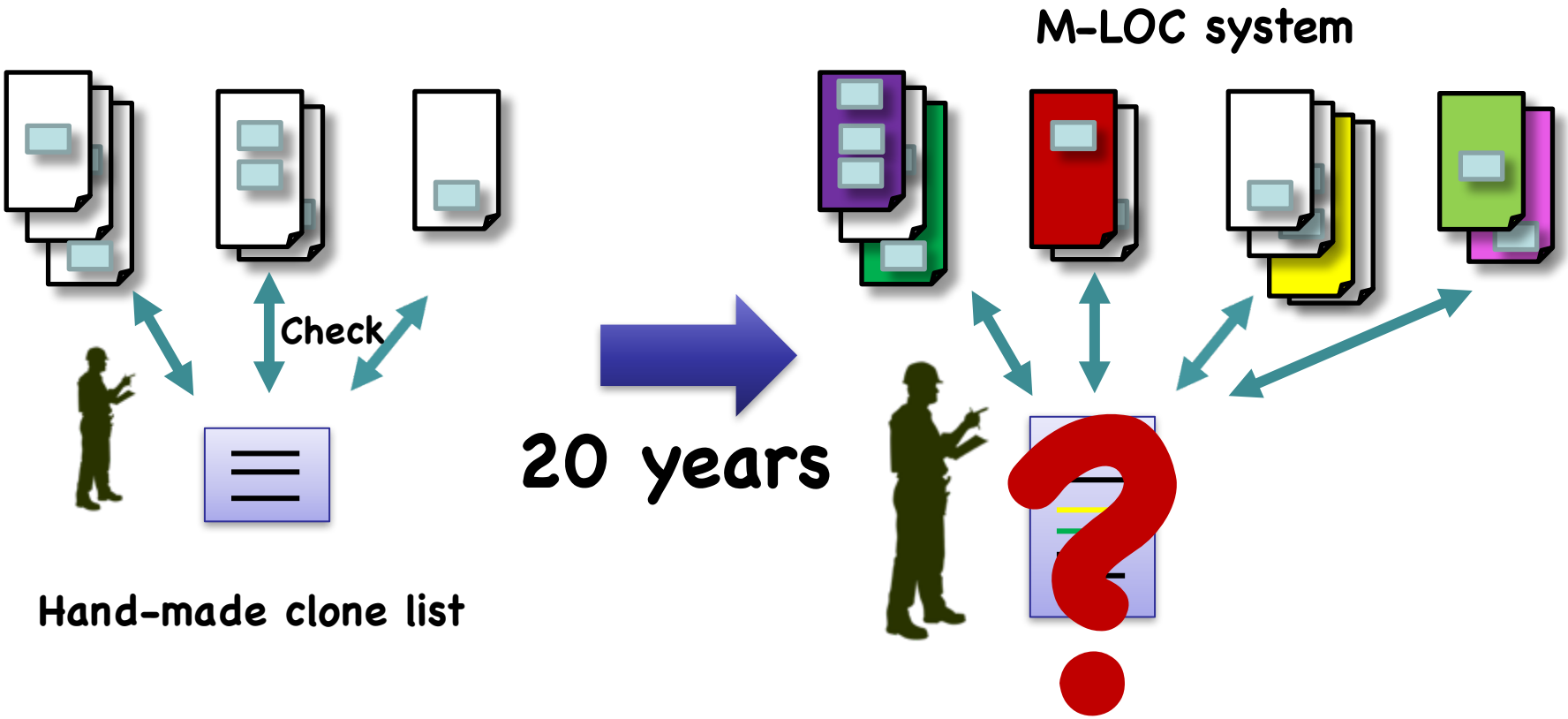
    e.g., `if(fp=fopen("file","r")==NULL){`
  - Intentionally: copy & paste practice



Opening

Issue at Company X

- Maintained code clones by hand



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.

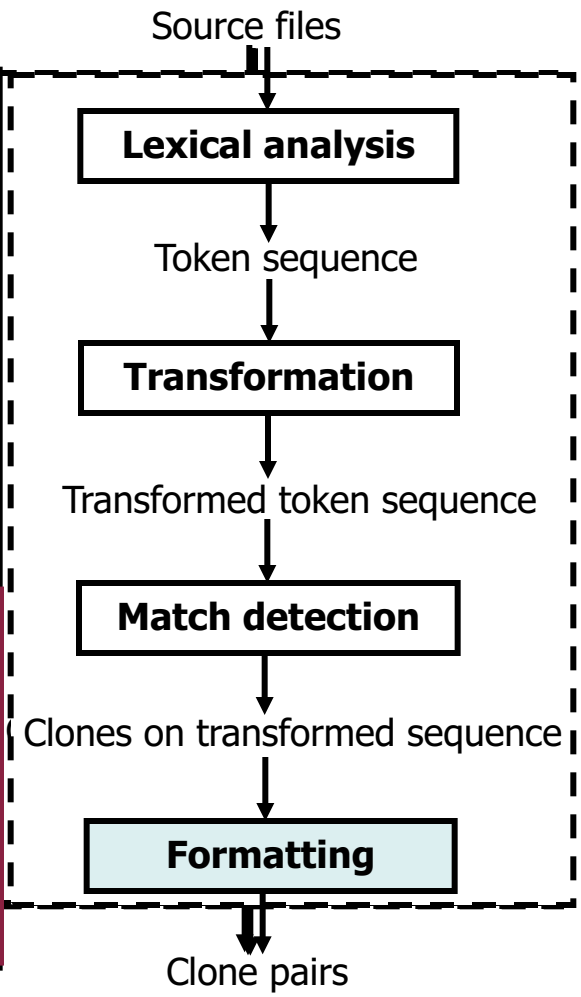
Example of Clones (type 2)

```
public static byte[] mergeLocalFileDataData(Zi
    final boolean lastIsUnparseableHolder = dat
        && data[data.length - 1] instanceof Un
    int regularExtraFieldCount =
        lastIsUnparseableHolder ? data.length -

    int sum = WORD * regularExtraFieldCount;
    for (ZipExtraField element : data) {
        sum += element.getLocalFileDataLength()
    }

    byte[] result = new byte[sum];
    int start = 0;
    for (int i = 0; i < regularExtraFieldCount;
        System.arraycopy(data[i].getHeaderId().
                         0, result, start, 2);
        System.arraycopy(data[i].getLocalFileDa
                         0, result, start + 2,
        byte[] local = data[i].getLocalFileData
        System.arraycopy(local, 0, result, sta
        start += (local.length + WORD);
    }
    if (lastIsUnparseableHolder) {
        byte[] local = data[data.length - 1].ge
        System.arraycopy(local, 0, result, sta
    }
    return result;
}
```

```
public static byte[] mergeCentralDirectoryData(ZipExtraFie
    final boolean lastIsUnparseableHolder = data.length >
        && data[data.length - 1] instanceof UnparseableExt
    int regularExtraFieldCount =
        lastIsUnparseableHolder ? data.length - 1 : data.l

    int sum = WORD * regularExtraFieldCount;
    for (ZipExtraField element : data) {
        sum += element.getCentralDirectoryLength().getValu
    }
    byte[] result = new byte[sum];
    int start = 0;
    for (int i = 0; i < regularExtraFieldCount; i++) {
        System.arraycopy(data[i].getHeaderId().getBytes(),
                         0, result, start, 2);
        System.arraycopy(data[i].getCentralDirectoryLength
                         0, result, start + 2, 2);
        byte[] local = data[i].getCentralDirectoryData();
        System.arraycopy(local, 0, result, start + WORD, 1
        start += (local.length + WORD);
    }
    if (lastIsUnparseableHolder) {
        byte[] local = data[data.length - 1].getCentralDir
        System.arraycopy(local, 0, result, start, local.le
    }
    return result;
}
```

A Clone Detection Process

```
1. static void foo() throws RESyntaxException {
2.    String a[] = new String [] { "123,400", "abc", "orange 100" };
3.    org.apache.regexp.RE pat = new org.apache.regexp.RE("[0-9,]+");
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. }
```

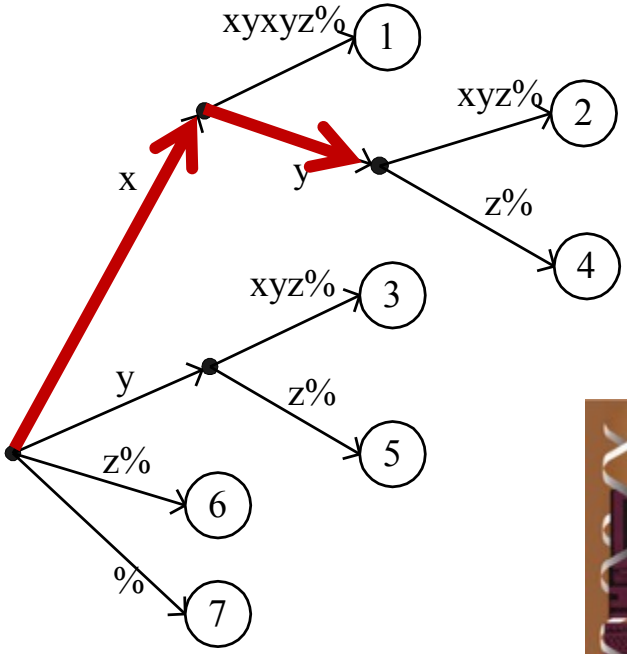


Match Detection Algorithms

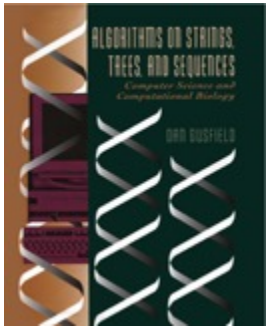
1	2	3	4	5	6	7
x	**x**	**y**	**x**	**y**	**z**	**%**

		1	2	3	4	5	6	7
		x	x	y	x	y	z	%
1	x	*	*		*			
2	x	*	*		*			
3	y			*		*		
4	x	*	*		*			
5	y			*		*		
6	z						*	
7	%							*

Matching Table



Suffix Tree



Initial Result

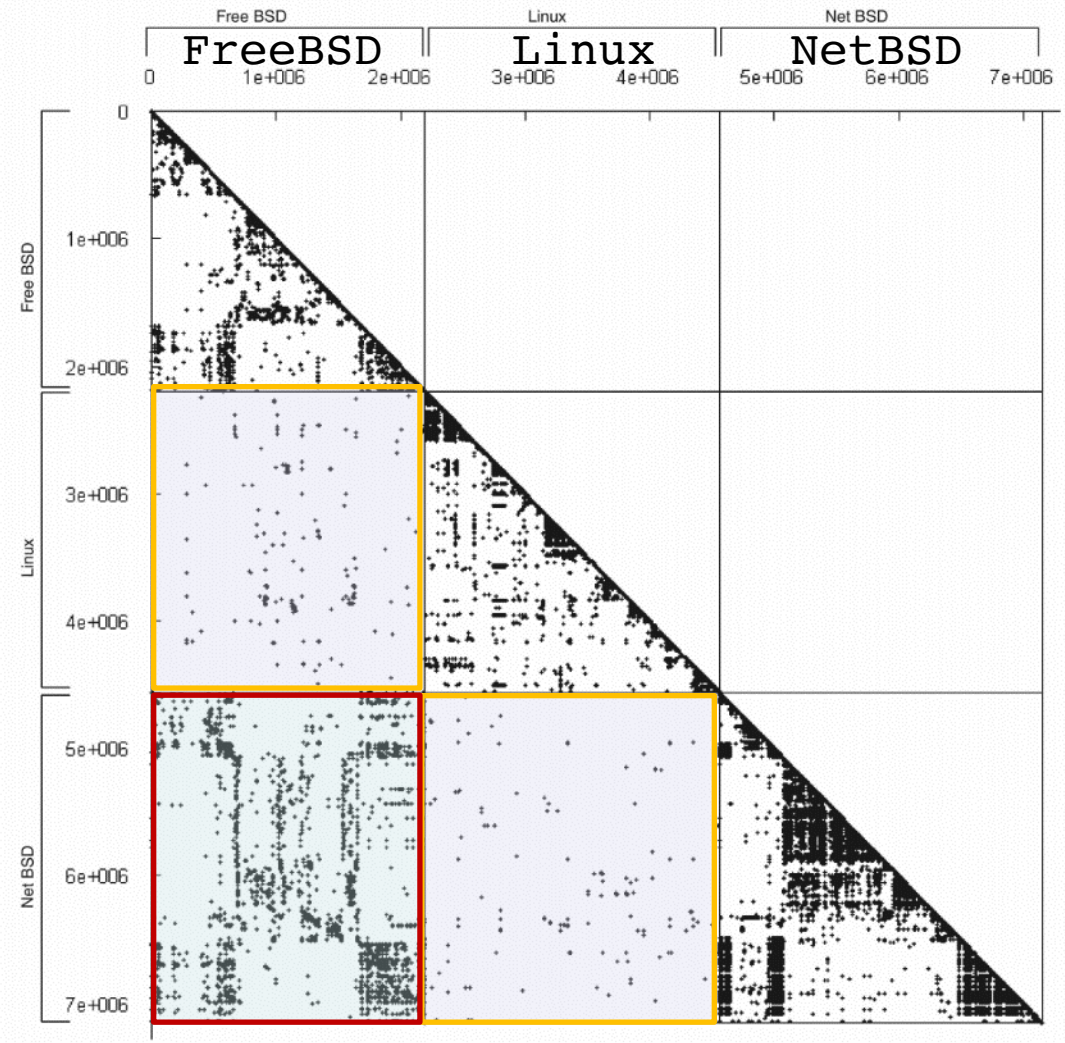
- Initial prototype had been implemented in a few weeks
- Limited scalability
  - Refined & tuned
  - Bought powerful workstation and expensive memory

Clones between Unix Kernels

FreeBSD

Linux

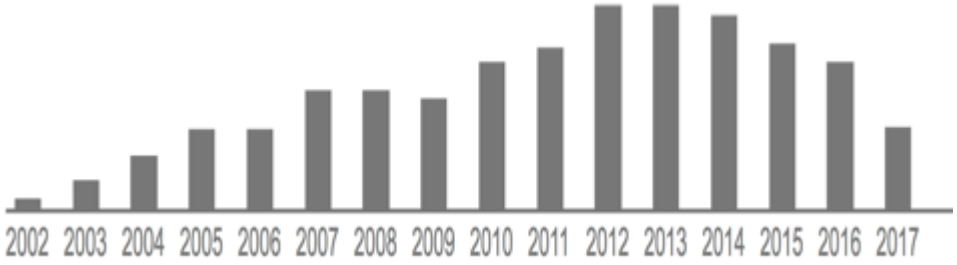
NetBSD



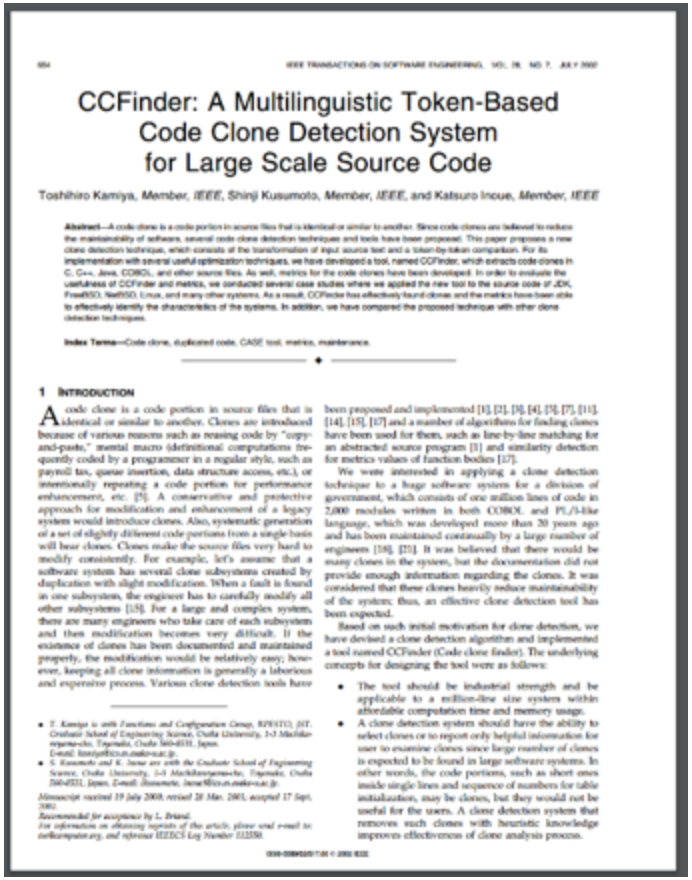
Let's Submit to IEEE TSE !

CCFinder: a multilinguistic token-based code clone detection system for large scale source code, T Kamiya, S Kusumoto, K Inoue, IEEE Transactions on Software Engineering, Vol.28, No.7, pp.654-670, 2002.

- 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



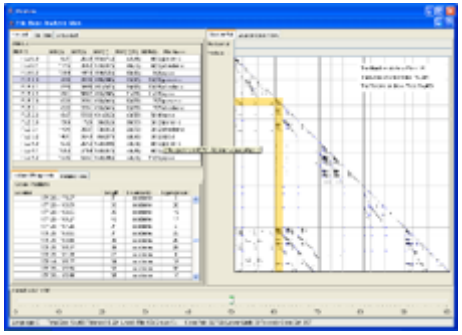
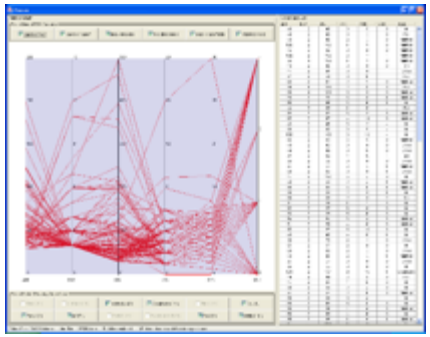
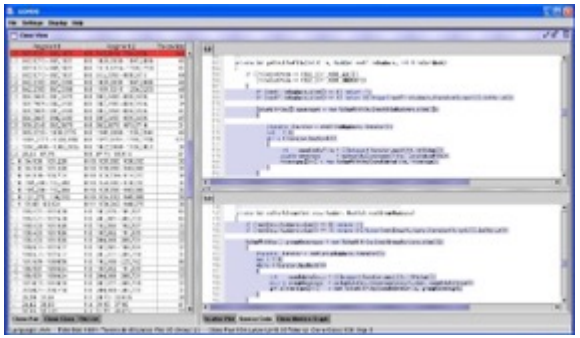
CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code

Toshihiro Kamiya, Member, IEEE, Shinji Kusumoto, Member, IEEE, and Katsuro Inoue, Member, IEEE

1 Introduction

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
Last update 18 Mar. 2005, Since 2005.
Toshihiro Kamiya

CCFinder

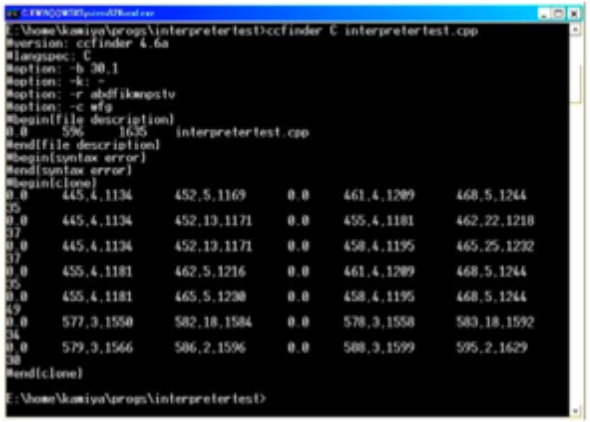
CCFinder is a tool for detecting code clones and has the following features:

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



> 5,000 downloads

CCFinder is implemented in C++, and currently has binary executable files for Windows98/Me/2000/XP. A paper by [Kamiya2002] 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.

Finding Duplicate Code by using Code Clone Detection

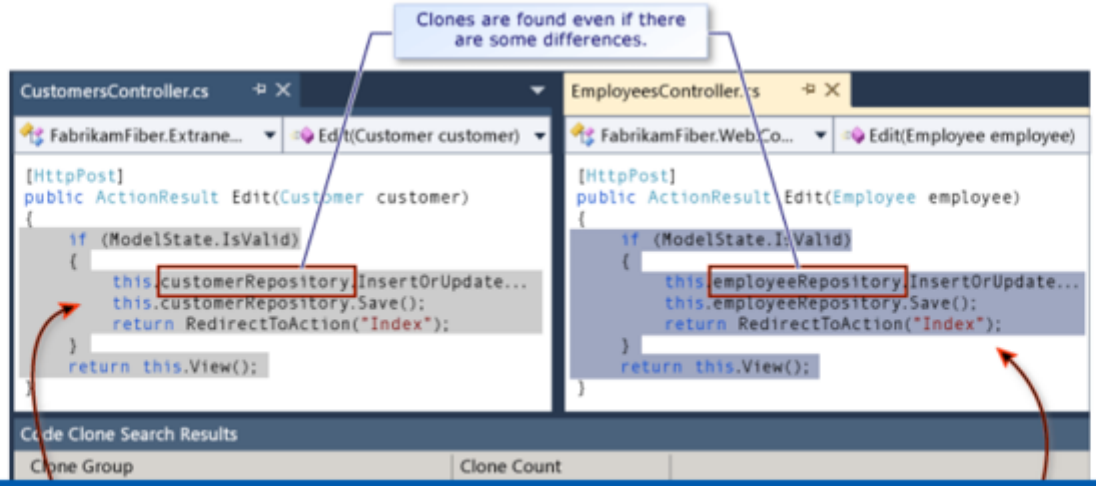
The new home for Visual Studio documentation is Visual Studio 2017 Documentation on docs.microsoft.com.

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.



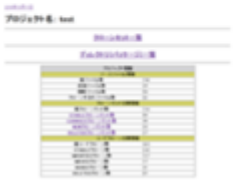
Clone Notifier

概要 ダウンロード 文書

[ス] 8月に発生した米駆逐艦とタンカーの衝突事故はユーザーインターフェイスも一因との指摘

プロジェクトの説明

レビューする Webページ 開発情報



画像一覧

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。 詳しい使用方法は https://osdn.net/projects/clonenotifier/downloads/66557/manual.pdf/ を参照ください。

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。

文書を見る

RSSを取得

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)

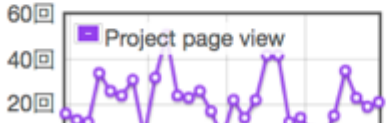
ダウンロードファイル一覧

レビュー

あなたの評価 レビューする

平均評価
5.0

5つ星 1
4つ星 0
3つ星 0



検索

いいね！ 0 ツイート G+

プロジェクト情報

最終更新
2016-10-12 15:28

登録日
2013-10-29 12:53

プロジェクトランキング
活発度順位: 圏外
ダウンロード順位: 圏外

開発メンバー
ejchoi, k-yokoi, nori_yos, y-yuuki

メンバー一覧

ソフトウェアマップ

ライセンス
MIT/X Consortium License
主要対話語
英語, 日本語
オペレーティングシステム
Windows
プログラミング言語
Java
トピック
ソースコード解析

関連プロジェクト

Experience of Finding Inconsistently-Changed Bugs in Code Clones of Mobile Software

Katsuro Inoue†, Yoshiki Higo†, Norihiro Yoshida†, Eunjong Choi†, Shinji Kusumoto†,
Kyonghwan Kim‡, Wonjin Park‡, and Eunha Lee‡
†*Osaka University* ‡*Samsung Electronics Co.*
Osaka, Japan *Suwon, South Korea*
{*inoue, higo, n-yosida, ejchoi, kusumoto*}*@ist.osaka-u.ac.jp*
{*kyonghwan73.kim, wj23.park, leeeunha*}*@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 country/area constraints, and so many code clones

We have restructured and modified various parts of the 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.

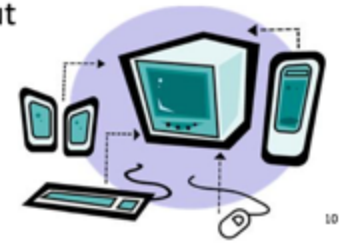
II. CloneInspector

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.

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



10

Applications

A

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

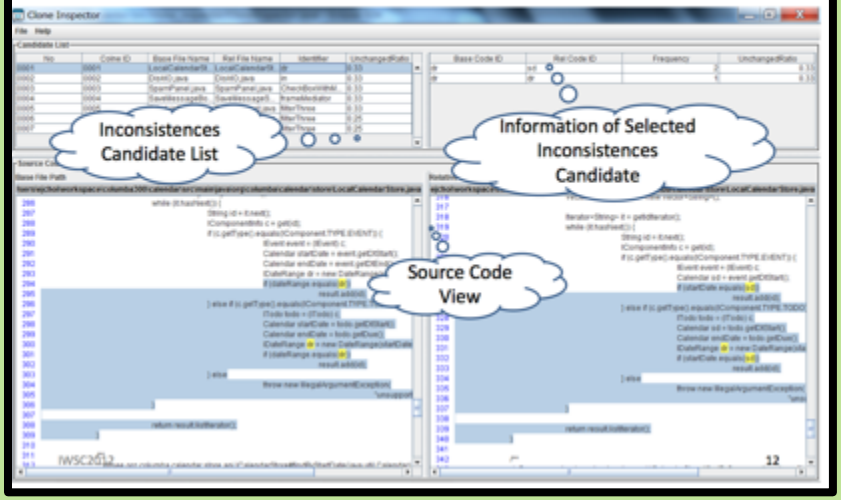
B

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

IWSC2012

11

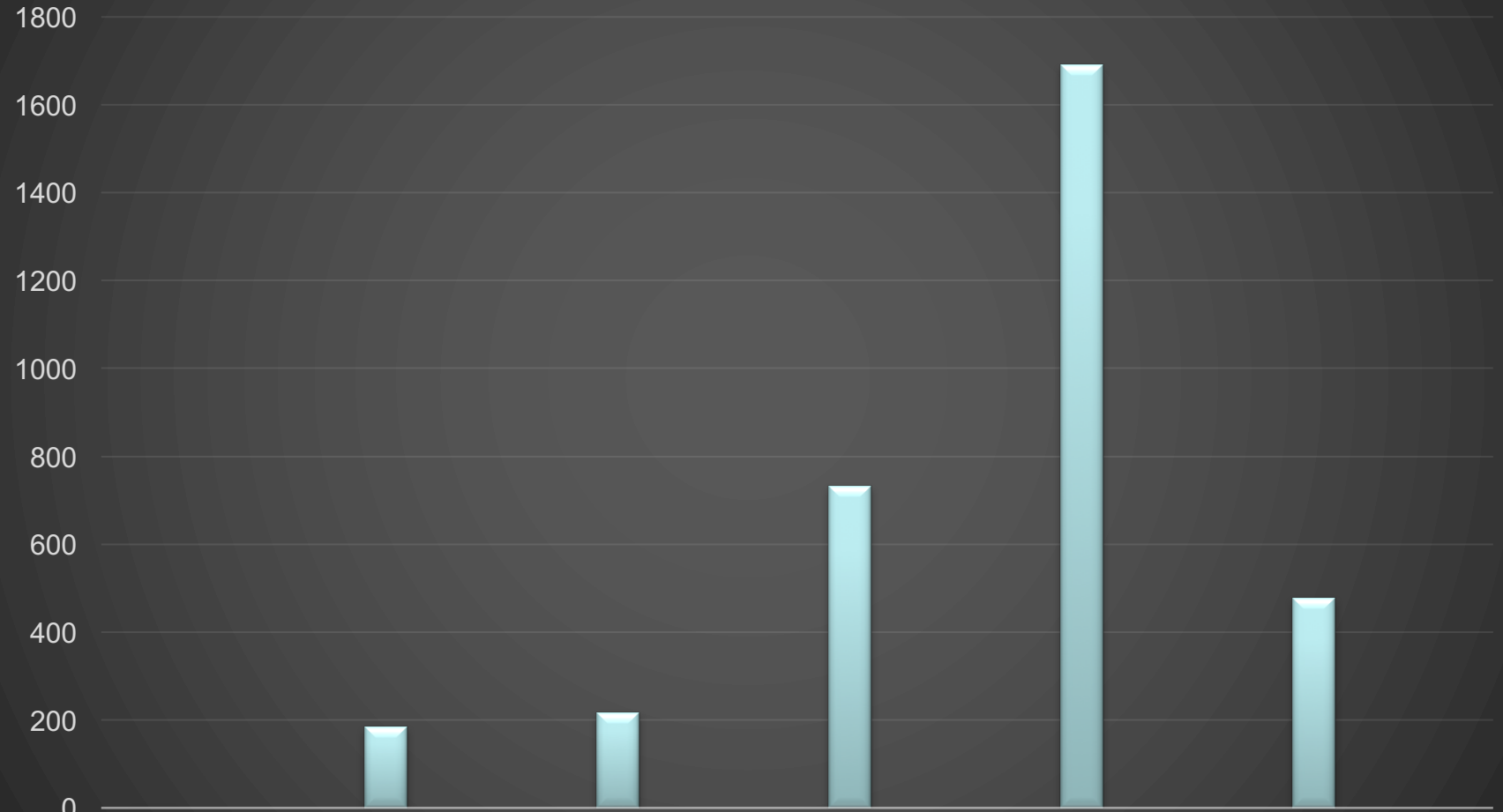
Snapshot of Clone Inspector



IWSC2012

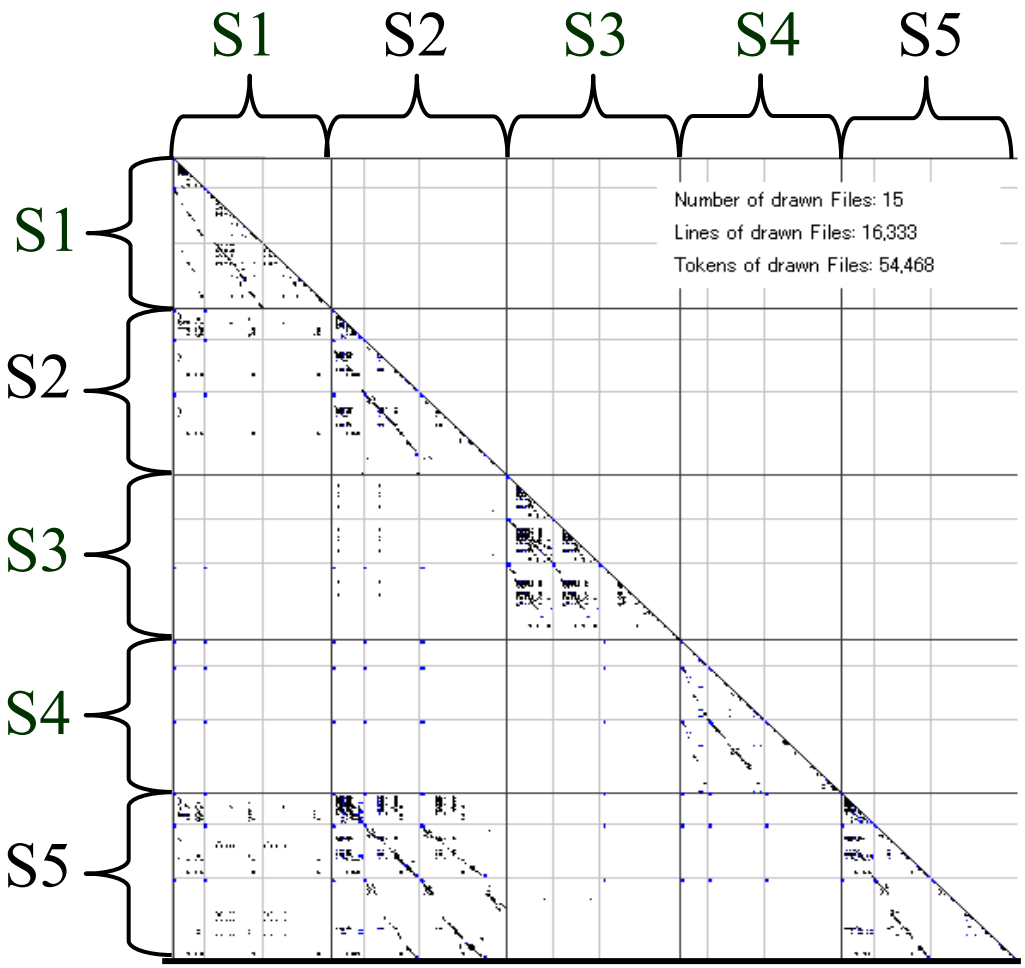
12

Papers Related to Code Clone

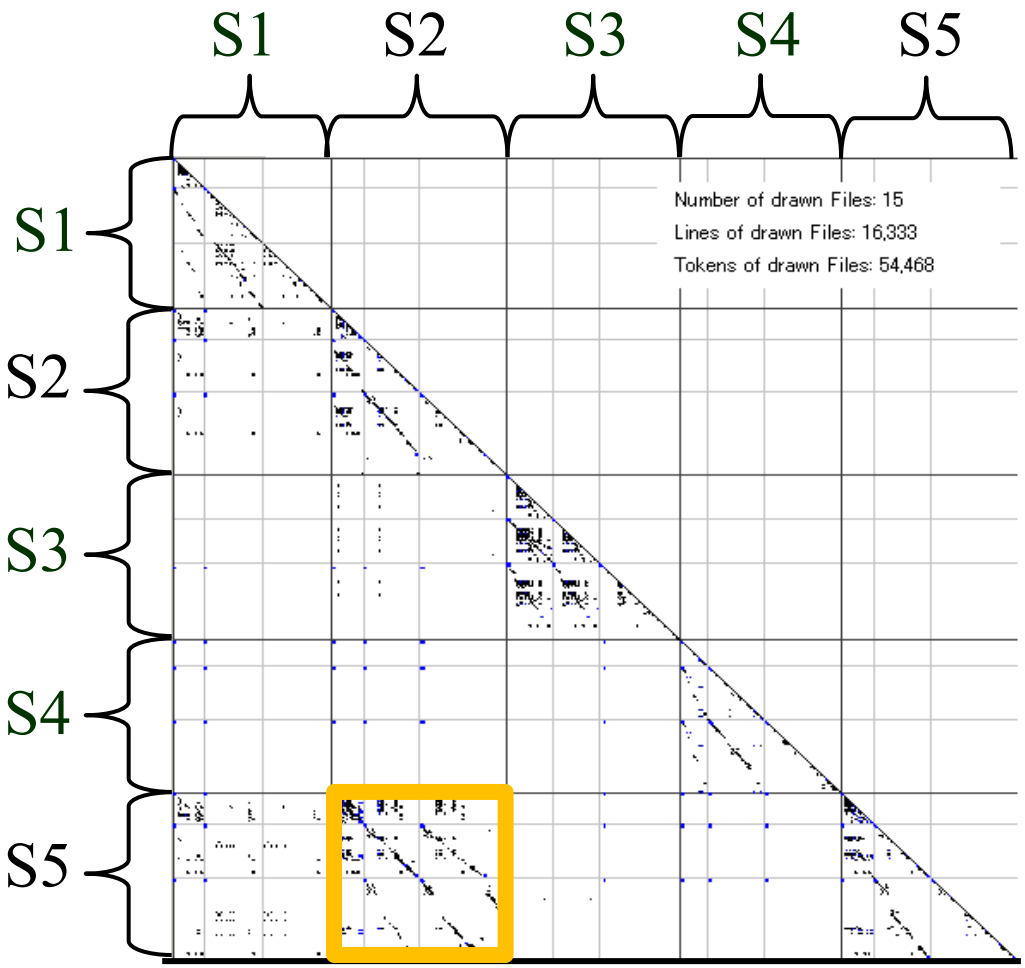


Applying to New Fields

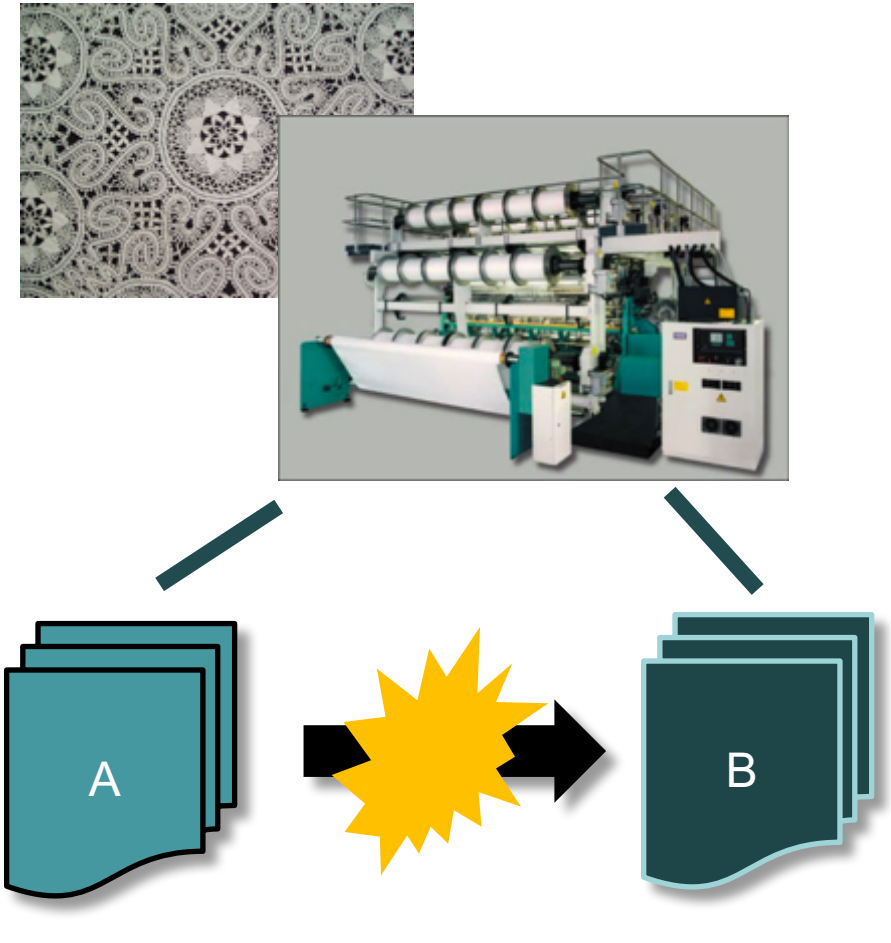
Compiler Construction Class



Compiler Construction Class



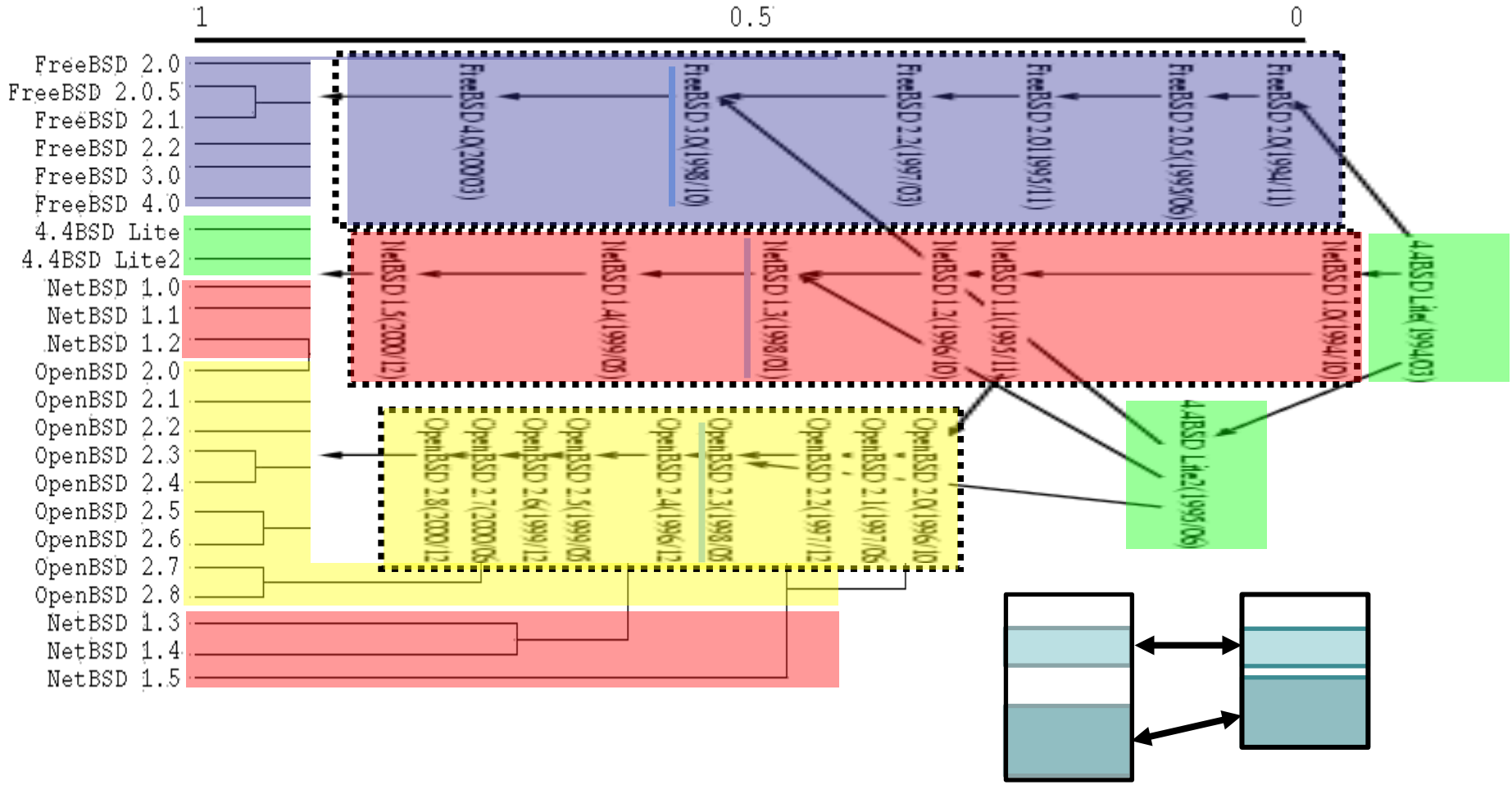
Law Suit of Lace-making Machines



平成９年(ワ)第１２４０２号　約定金請求事件
口頭弁論終結日　　平成１３年７月１８日
判　　決
原　　告　　株式会社高電
訴訟代理人弁護士　　片　山　善
同　　岡　村　久
被　　告　　株式会社ジェ
ケムラ
訴訟代理人弁護士　　坂　田
同　　草　地　邦
主　　文
１　被告は、原告に対し、金１５５０万１５００円
９年１２月１７日から支払済みまで年６分の割合による金員
２　原告のその余の請求を棄却する。
３　訴訟費用はこれを２分し、その１を原告の、そ
る。
４　この判決は、第１項に限り、仮に執行すること
事実及び理由
第１　請求
被告は、原告に対し、金３５００万円及びこれに対す
日（訴状送達の日の翌日）から支払済みまで年６分の割合に
第２　事案の概要
１　前提事実（末尾に証拠の掲記のない事実は当事者間に
(1)　原告は、コンピュータ機器の販売、コンピュータ
売等を目的とする株式会社であり、被告は、織物用電子機器
とする株式会社である。
(2)　原告は、平成３年４月１日、被告との間で、次の
（ソフトウエア）取引契約（以下「本件契約」という。）を
①　甲（被告）は乙（原告）の開発する製品を販売し
甲に販売する（第１条）。
②　対象とする本契約における製品は、織布情報作成
（専用電子計算機により編機を作動させるための
フトウエアに限り、これに用いる専用の入出力装置、光学式
フィック処理装置、または補助記憶装置は含まない。）（第
③　乙は甲に製品のソースプログラムを開示する。甲
ログラムを元に変更を加えた製品を顧客に販売することがで
ただし甲は開示されたソースプログラムを他社に
ない（第３条）。

Clones in near systems

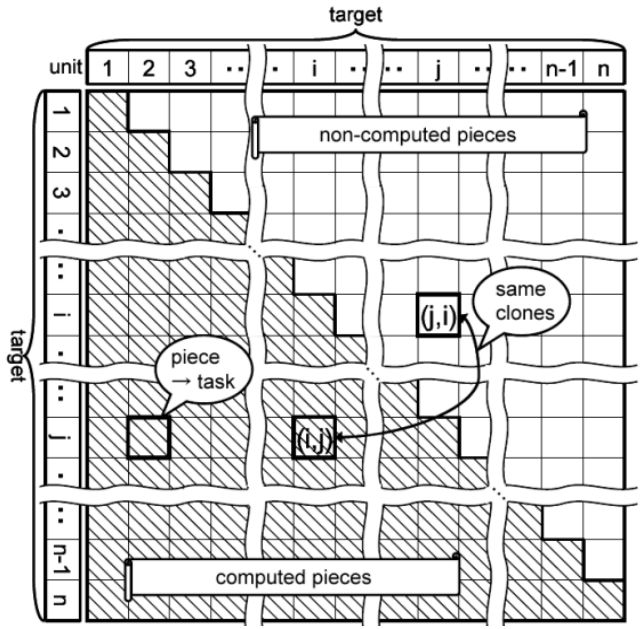
System Evolution Analysis
— BSD Unix Clustering —



Similarity Measure ← Cover Ratio

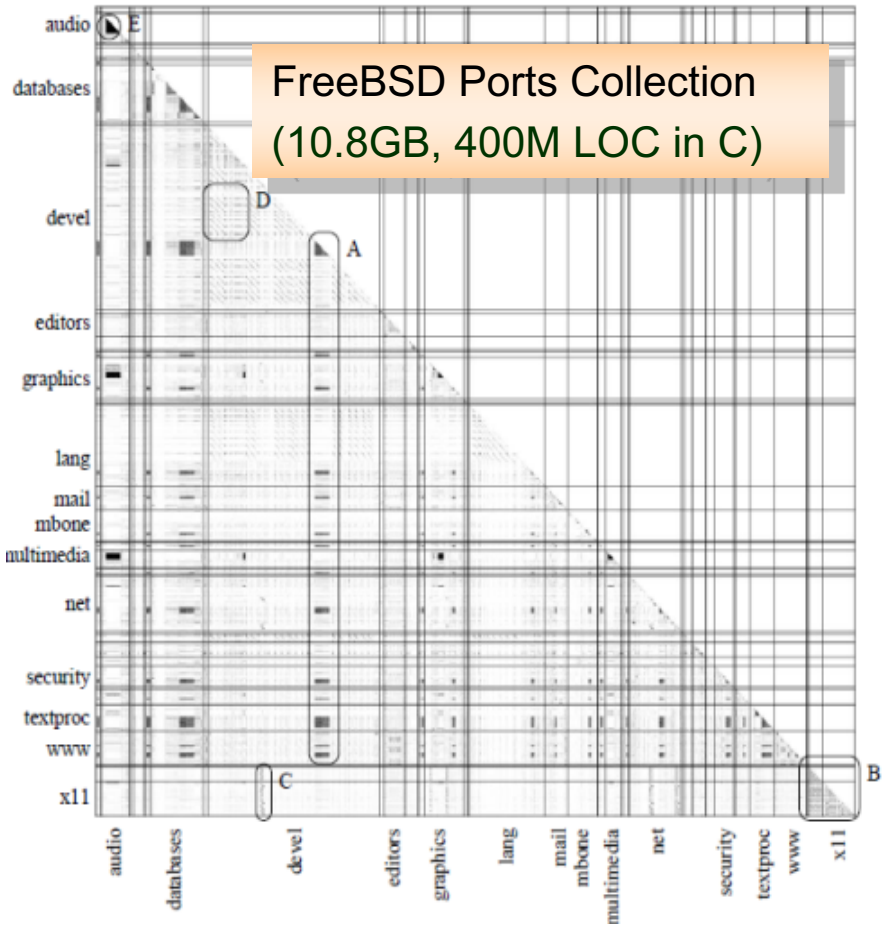
Chasing Scalability

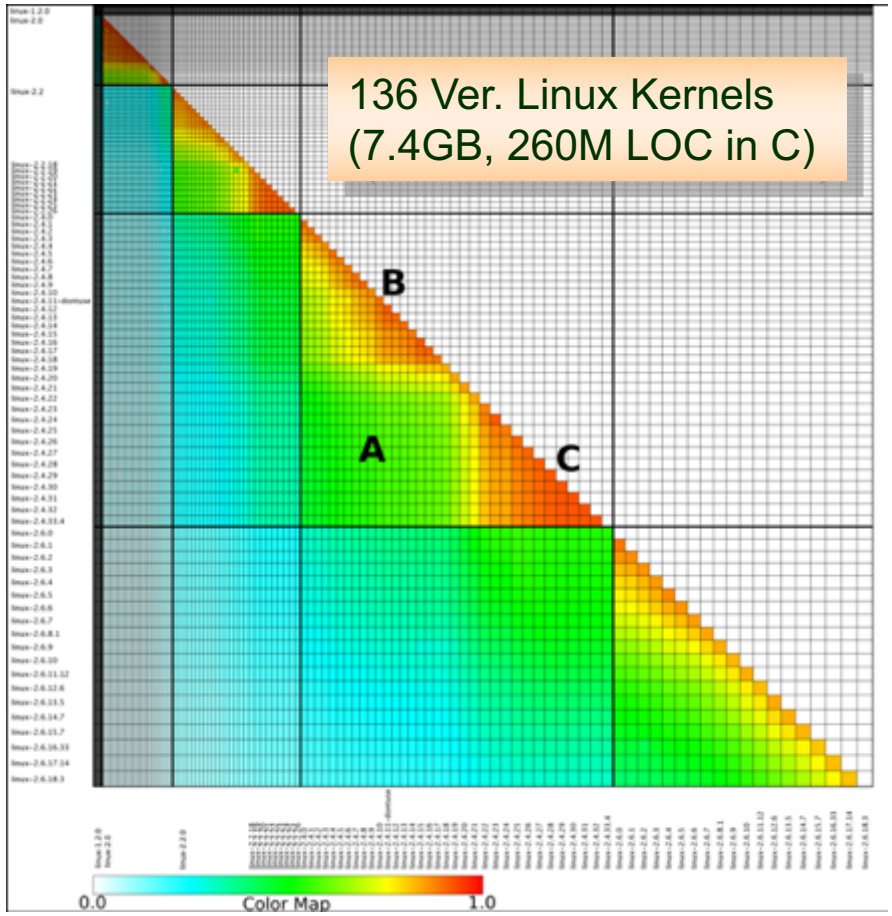
- CCFinder input limitation: A few M LOC due to the suffix tree algorithm --- on core
- Code clone detection is embarrassingly parallel problem --- divide and concur



Distributed CCFinder with 80 lab machines

FreeBSD Ports Collection / 136 Versions of Linux Kernel





Livieri, S., Higo, Y., Matsushita, M., Inoue, K., "Very-Large Scale Code Clone Analysis and Visualization of Open Source Programs Using Distributed CCFinder: D-CCFinder", ICSE2007.

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 descendents 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 [29] 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.

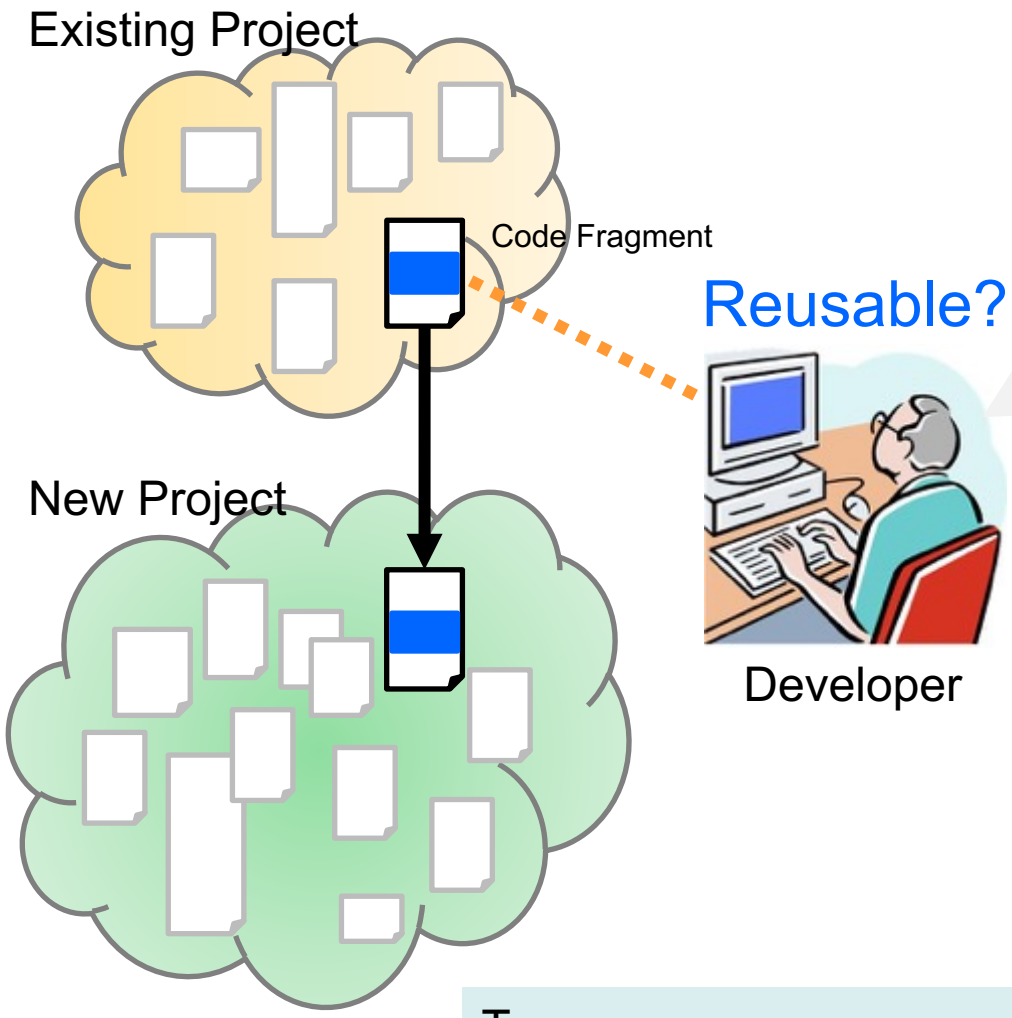
Code search engines such as Google Code Search [10] and Koders [3] are very useful tools to explore open source repositories for the origin and evolution of code. However, current code search engines only allows to get keywords and/or code attributes as their inputs, and they return source code files 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

34th International Conference on Software Engineering, pp.331-341, Zurich, Switzerland, 2012-05.

Developer's Concerns

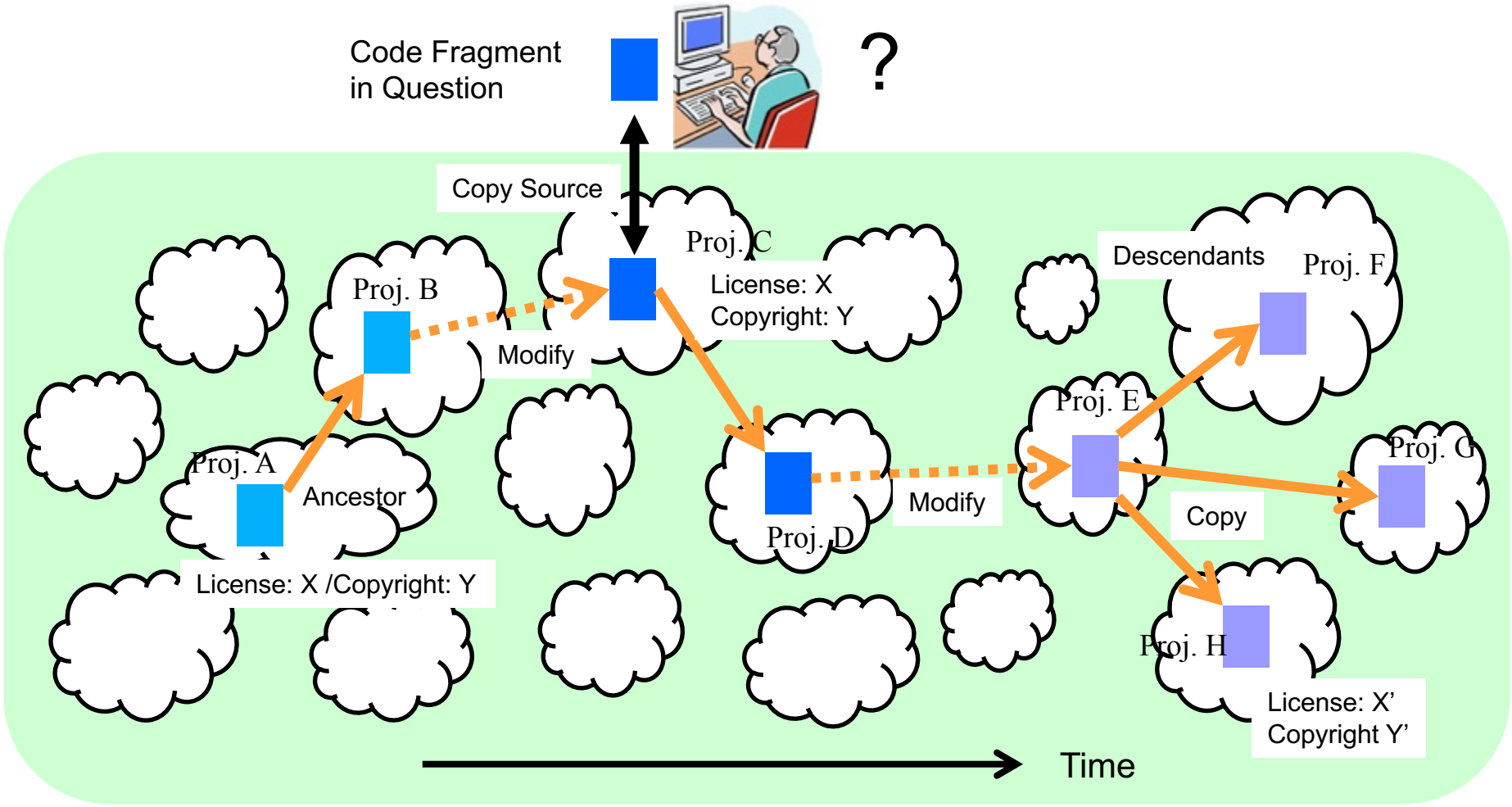


Concerns

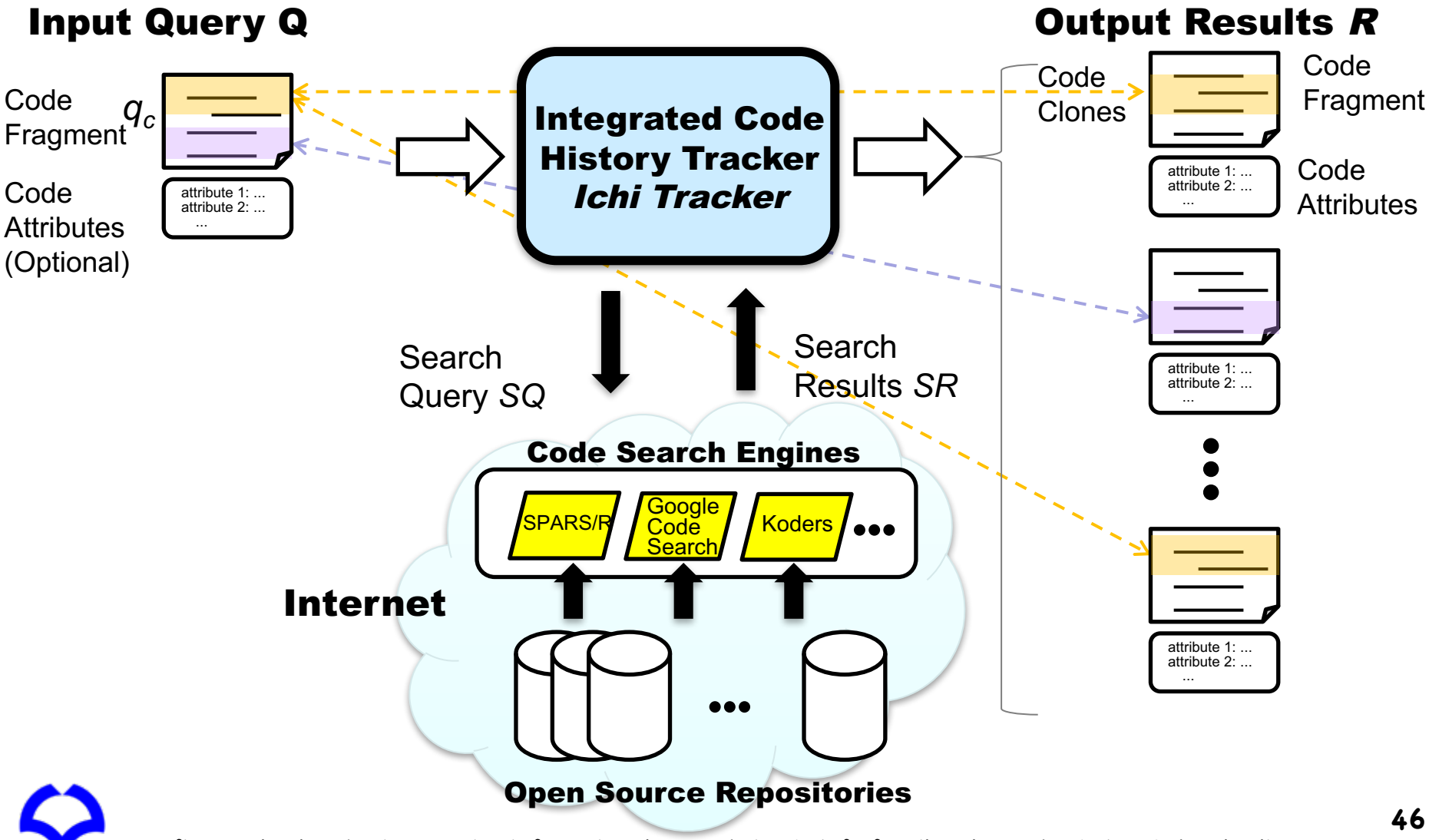
- Origin
  - Who?
  - When?
  - License?
  - Copyright?
- Evolution
  - Maintenance?
  - Popularity?
  - Newer version?
  - …

To ease concerns, a support system is needed

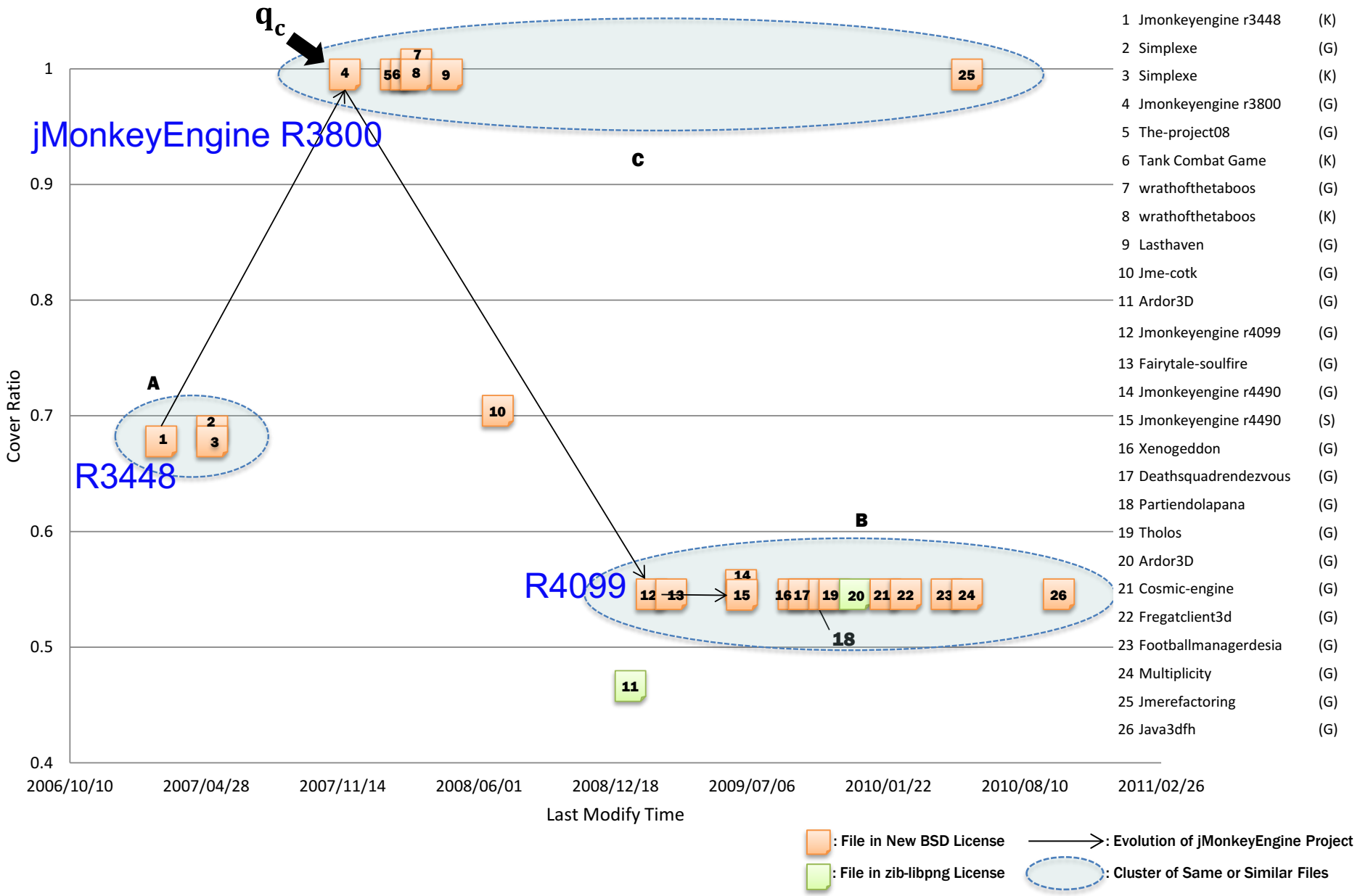
Code History Tracking System



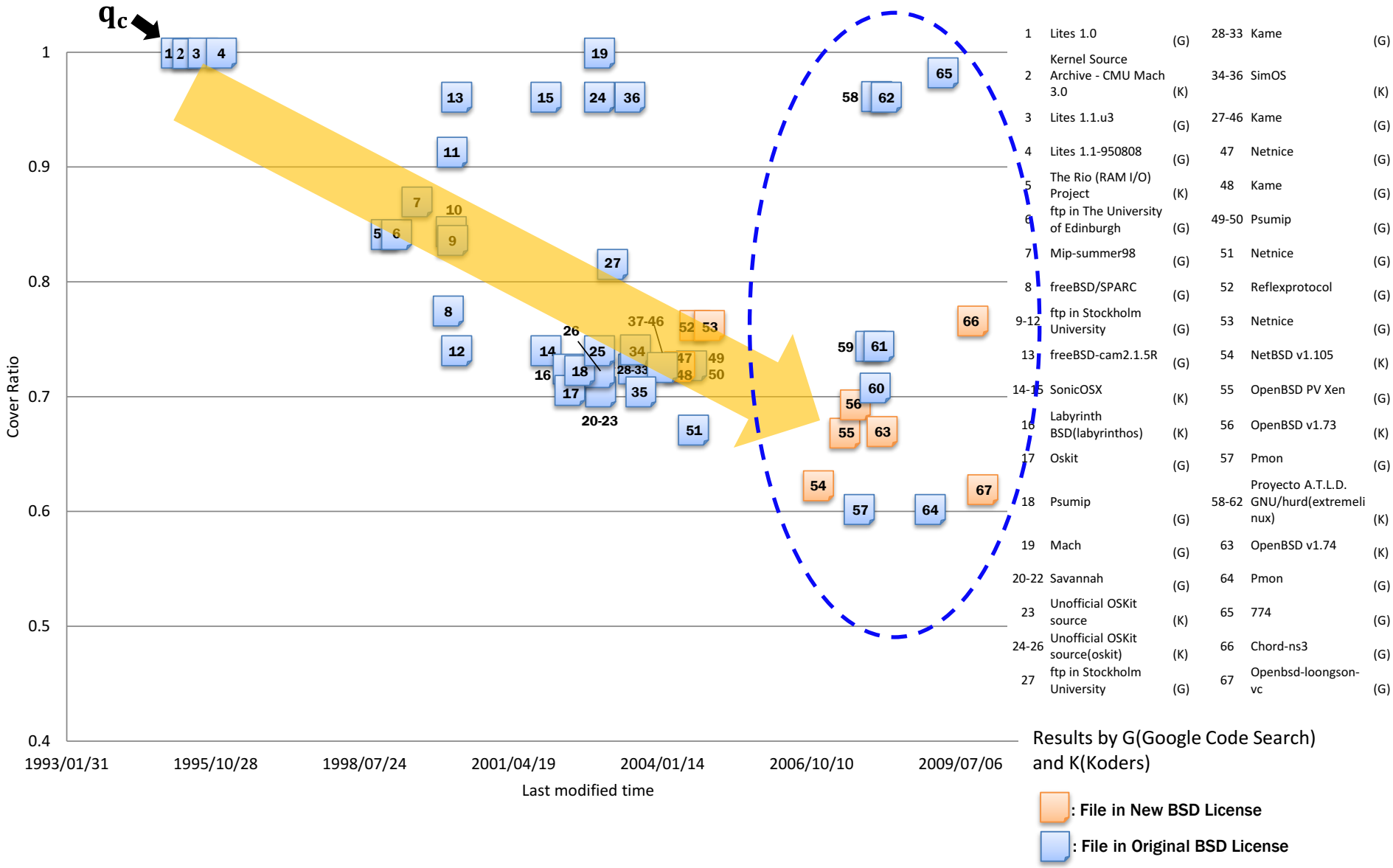
System Overview



Evolution Pattern of Texture.java

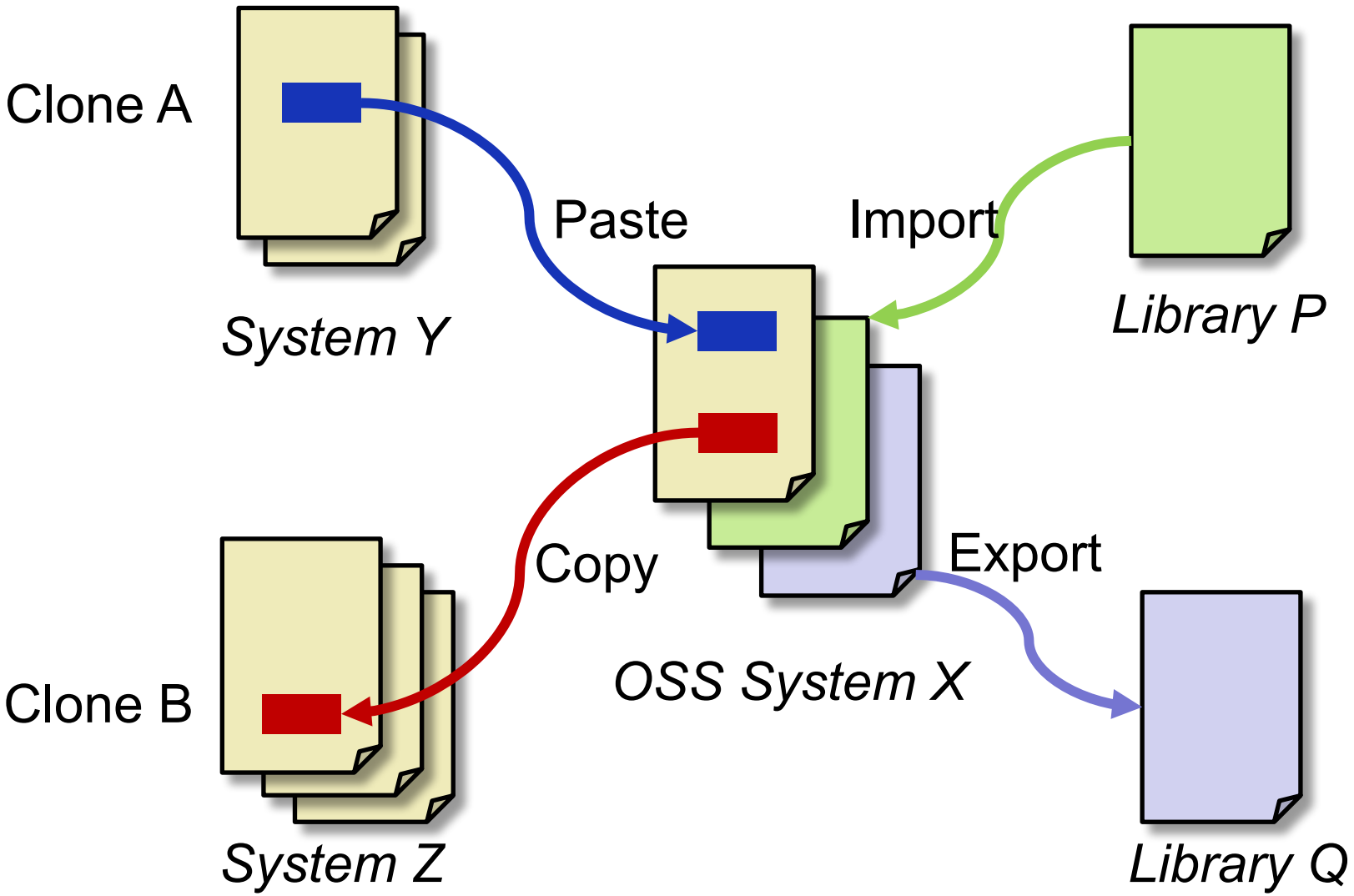


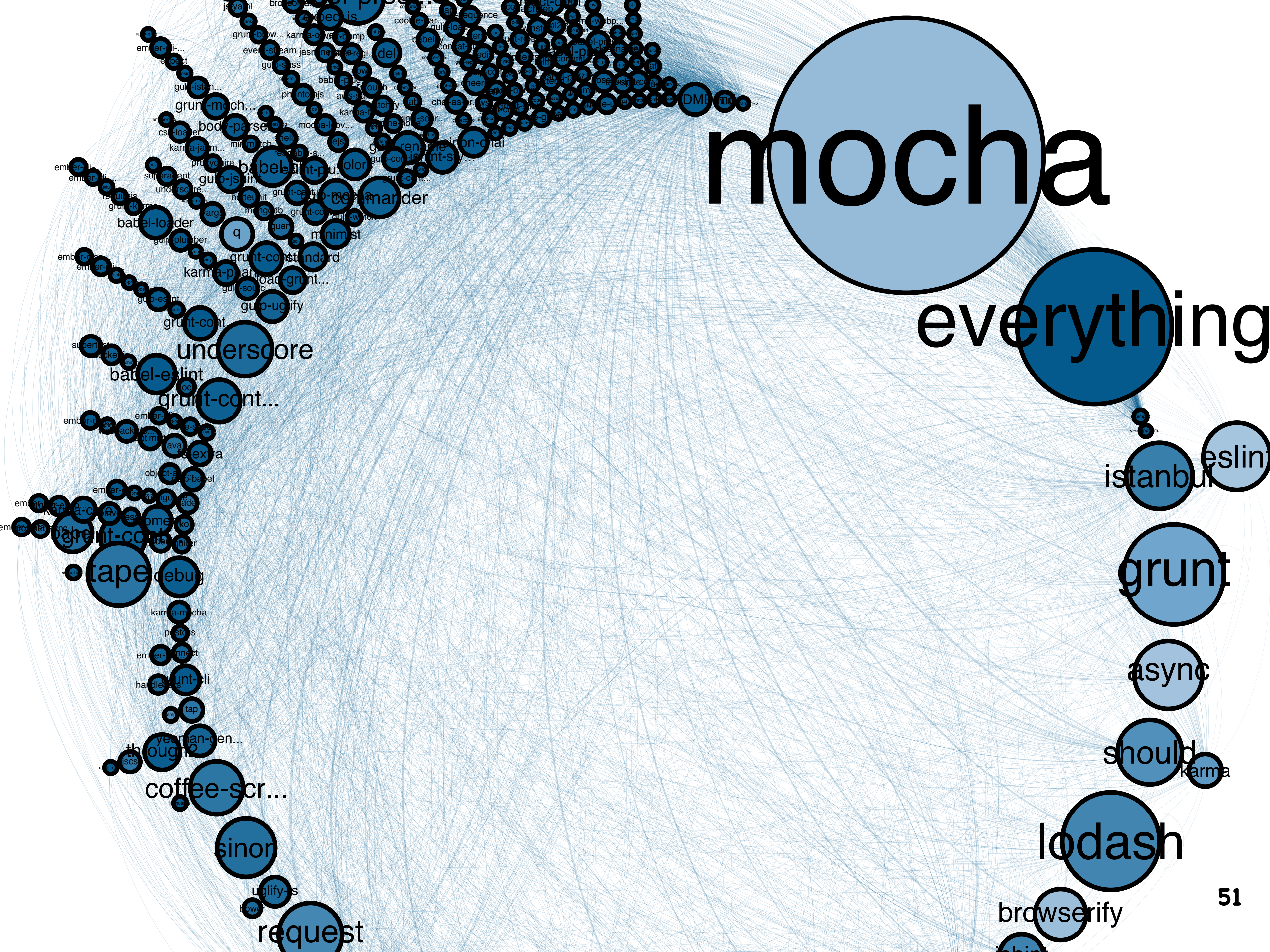
Evolution Pattern of kern_malloc

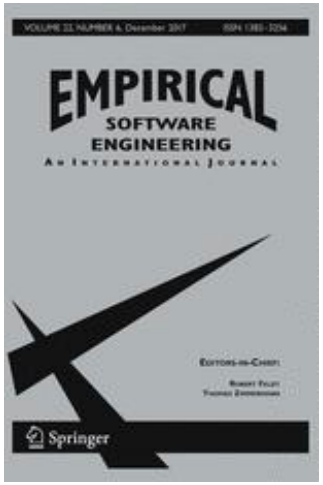


Current and Future

OSS Dependency




mocha
everything
eslint
istanbul
grunt
async
should
karma
lodash
browserify
request
sinon
coffee-scr...
tape
debug
underscore
grunt-cont...
babel-eslint
commander
q
minimist
standard
babel-loader
gulp-uglify
karma-mocha
uglify-js
yeoman-gen...
tap
phantomjs
sinon-chai



Empir Software Eng
DOI 10.1007/s10664-017-9521-5

CrossMark

Do developers update their library dependencies?

An empirical study on the impact of security advisories on library migration

Raula Gaikovina Kula[3] · Daniel M. German[2] · Ali Ouni[1,4] · Takashi Ishio[3] · Katsuro Inoue[1]

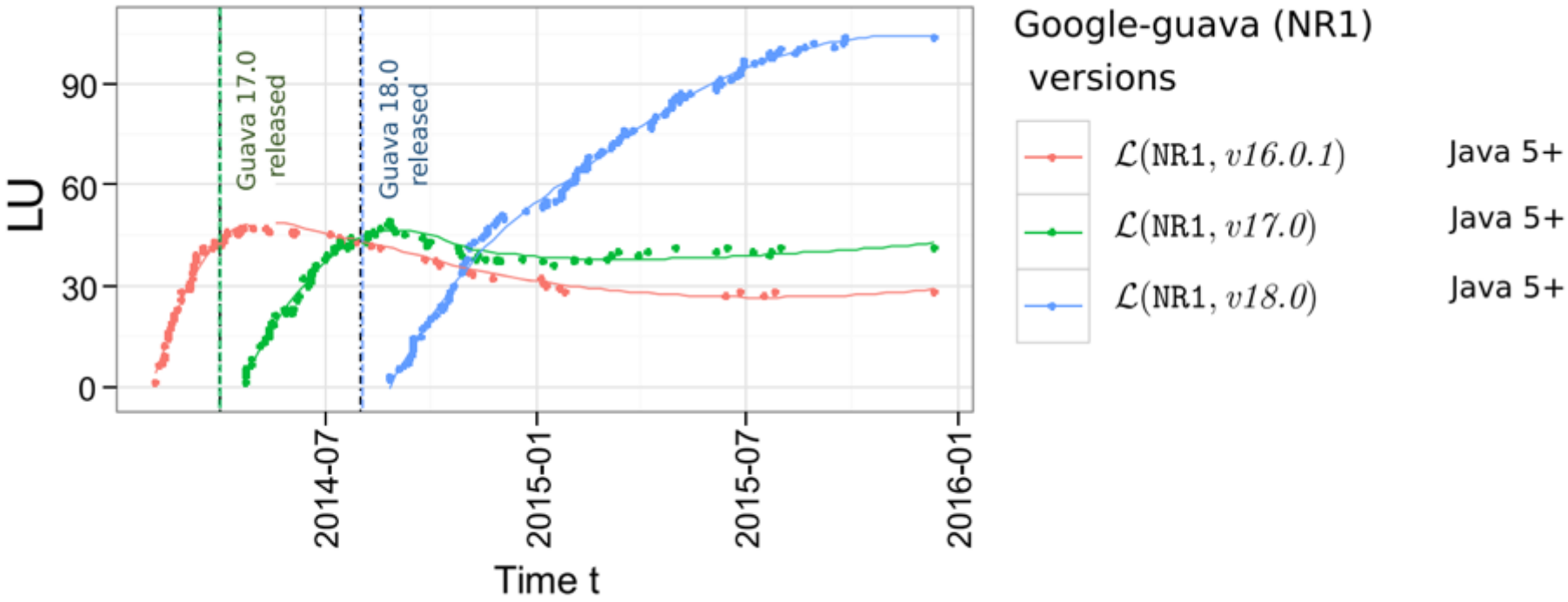
Abstract Third-party library reuse has become common practice in contemporary software development, as it includes several benefits for developers. Library dependencies are constantly evolving, with newly added features and patches that fix bugs in older versions. To take full advantage of third-party 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

Communicated by: Martin Robillard

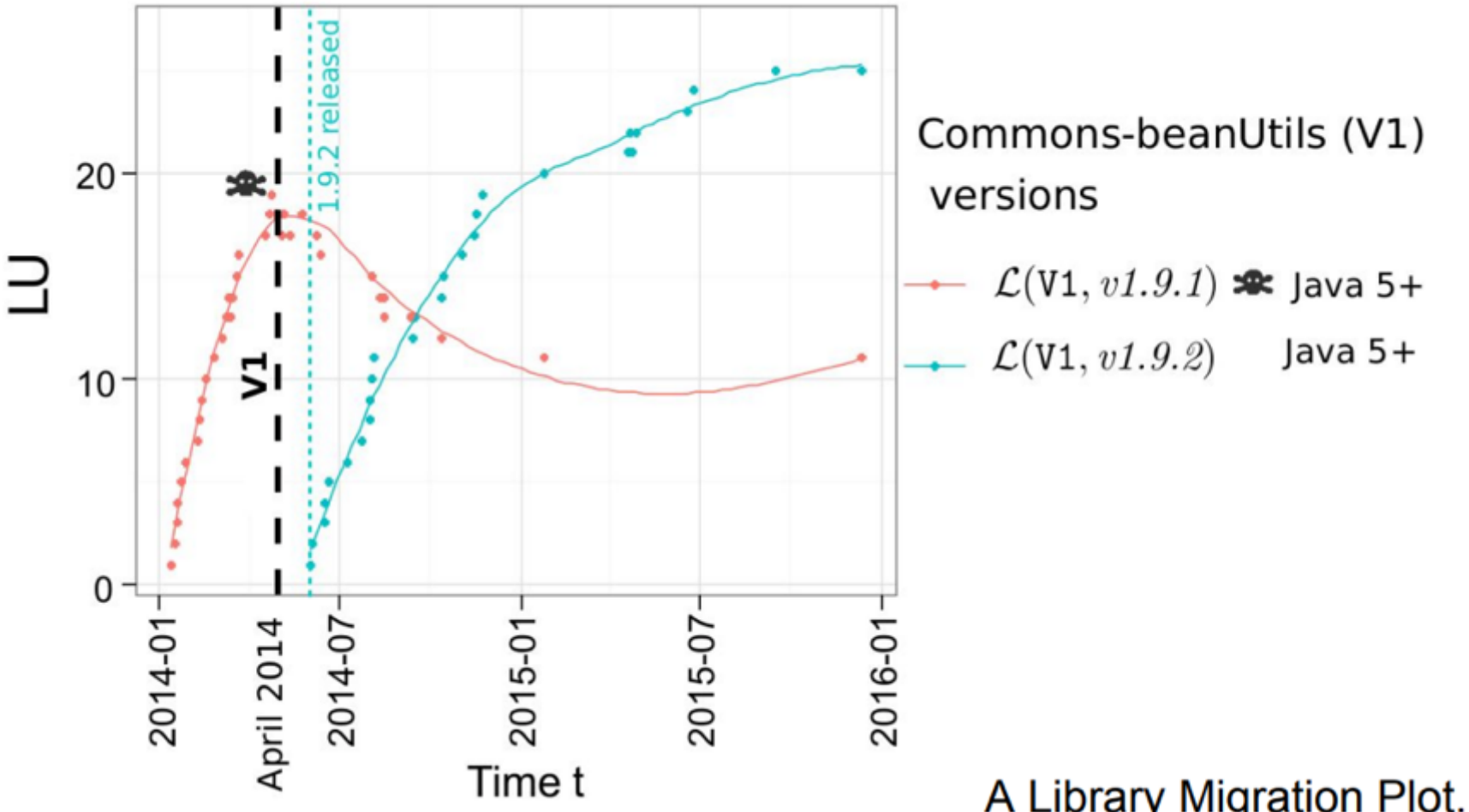
✉ Raula Gaikovina Kula
raula-k@is.naist.jp

Daniel M. German

Raula Gaikovina Kula, Daniel M. German, Ali Ouni, Takashi Ishio, Katsuro Inoue, Do developers update their library dependencies? An empirical study on the impact of security advisories on library migration, EMSE accepted & FSE2017 J. 1st.

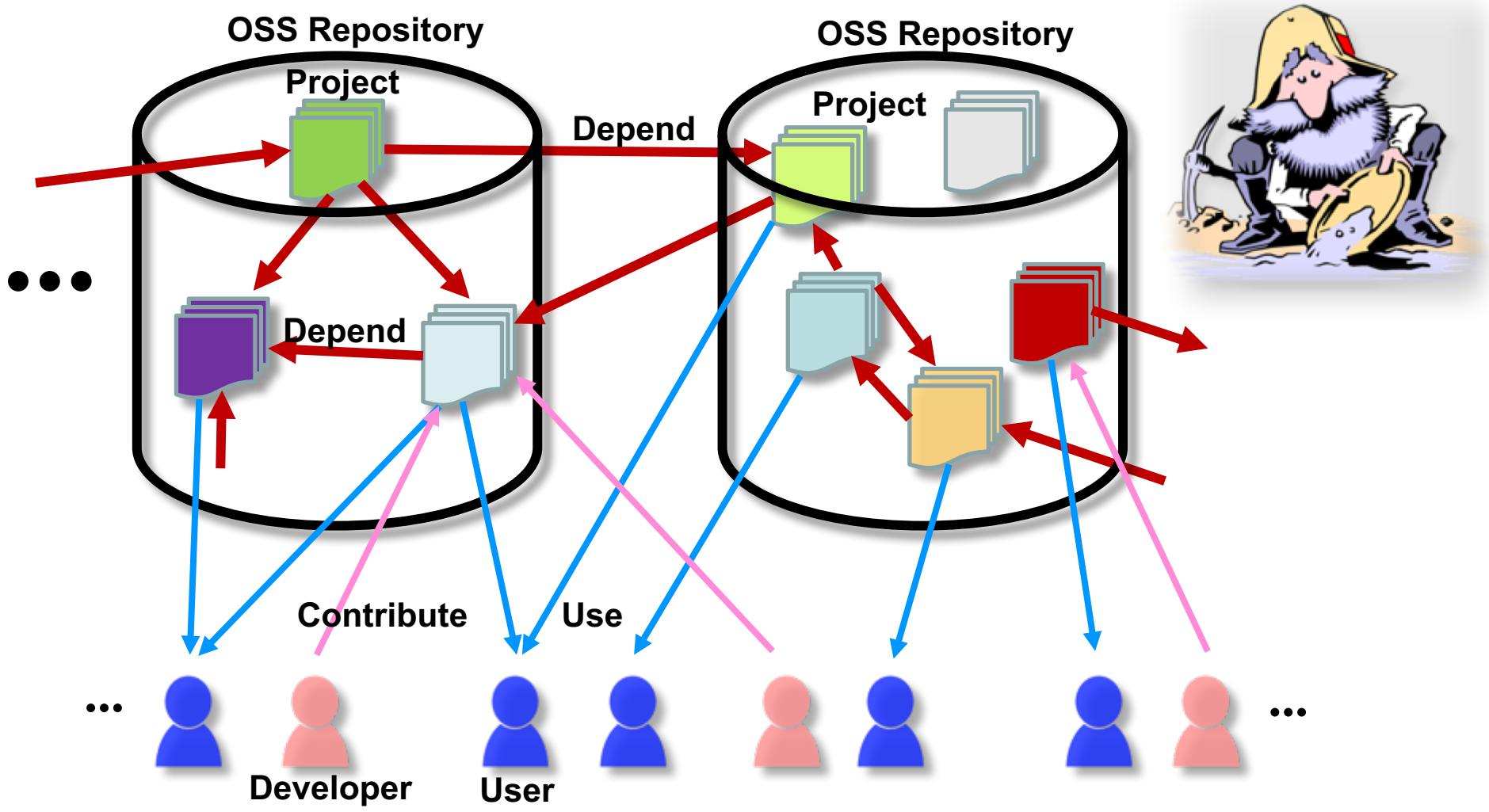


(a) LMP for consecutive releases of the `google-guava` (NR1) library



A Library Migration Plot.

OSS Ecosystem



Summary

Finding similar code in softwlare matters

Code clone analysis became very popular SE technology

Analyzing code similarity is key to know OSS provenance and evolution

Mining complex OSS dependencies

Exploring OSS universe with code similarity analysis

Thank you