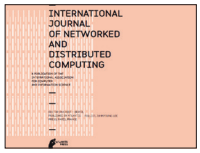
ATLANTIS PRESS

International Journal of Networked and Distributed Computing
Vol. **6**(**4**); December (2018), *pp.* 224–231
DOI: 10.2991/ijndc.2018.6.4.5; ISSN 2211-7946
https://www.atlantis-press.com/journals/ijndc



Improvement of the Dynamic Software Birthmark Process by Reducing the Time of the Extraction

Takanori Yokoi[1,*], Haruaki Tamada[2]

[1]*Division of Frontier Informatics, Graduate School of Kyoto Sangyo University, Motoyama, Kamigamo, Kita-ku, Kyoto, Kyoto 603-8555, Japan*
[2]*Faculty of Information Science and Engineering, Kyoto Sangyo University, Motoyama, Kamigamo, Kita-ku, Kyoto, Kyoto 603-8555, Japan*

ARTICLE INFO

Article History

Received 27 June 2018
Accepted 24 September 2018

Keywords

Dynamic birthmarks
unit tests
plagiarism detection
software protection

ABSTRACT

It is a quite tough task to detect the stolen programs since there is a quite huge number of programs in the world. The dynamic software birthmarks were proposed to detect the suspects of plagiarisms based on the runtime behavior of the programs. The detection process with the dynamic birthmarks is composed of extraction, and comparison phases. However, the extraction phase spends much time because it requires to prepare the inputs for running the programs. Generally, preparing the inputs requires the understanding about the target programs. Hence, this paper tries to reduce the extraction time without the understanding the programs by using the unit tests. We evaluated the credibility and resilience of properties of the dynamic birthmarks extracted by the proposed method. As a result, the similarities were greater than 0.8 among the newest two versions of the same products. On the other hand, similarities between different projects were under 0.355.

1. INTRODUCTION

The dynamic software birthmark methods are the methods to detect the stolen programs based on the runtime behavior of the target programs. For this, the dynamic birthmark methods extract the characteristics of programs themselves by running the target programs. Then, the extracted characteristics are compared and calculated the similarities between two characteristics. If the similarity is greater than the given threshold, either of programs is suspected the stolen program from the other. Based on the process, many researchers proposed various dynamic software birthmark methods focused on different aspect of the runtime behaviors. However, the problem is that above detection procedure consumes much time cost. The main cause is the extraction requires the suitable inputs, and the cost for preparing inputs is quite high. Because, the preparation of inputs need the understanding the target programs, and the dynamic birthmarks generally require several inputs for reliable results.

If script kiddies try to steal the programs, they simply copy the programs, and conceal the source codes. However, it is a quite tough task to detect them. First, quite a huge number of programs are released in the whole world. Second, we cannot use the source code for detection, and the binaries are sensitively changed by the compile options. Therefore, the software birthmarks are applied, however, the dynamic software birthmarks need the knowledge to apply them described above. Hence, this paper tries to reduce the required knowledge to apply the dynamic software birthmarks for detecting software theft. For this, we eliminate on the understanding cost for extracting the dynamic software birthmarks, by using the unit test codes. Generally, the unit test codes are aims to find the bugs of the programs, they give the inputs to the target programs. We focus on the inputs in the unit test codes for extracting the dynamic software birthmarks. That is, the proposed method can extract the dynamic software birthmarks without any understanding of the target programs.

The rest of this paper is organized as follows. Section 2 gives the general definition about the birthmarks. Section 3 describes the proposed method. Section 4 shows the experimental evaluations and their results. Finally, Section 5 concludes the paper.

2. PRELIMINARY

2.1. Definition of the Birthmarks

Before describing the proposed method, we explain the definition of software birthmarks. The software birthmarks are defined by Tamada et al. [1,2]. Based on the definition, the dynamic birthmarks are defined as follows by Myles et al. [3] and Tamada et al. [4].

Definition 1. (Dynamic Software Birthmark) Let p and q be given programs. Let I be given input for p or q. Further, let $B(p, I)$ be a set of characteristics extracted from runtime information obtained from providing p with I by a certain method B. If the conditions below are met, $B(p, I)$ is said to be a dynamic birthmark of p with I.

Condition 1: *$B(p, I)$ is obtained from providing a program p with an input I.*

* *Corresponding author. Email: i1788287@cc.kyoto-su.ac.jp*

Condition 2: *If q is a copy of p, then B(p, I) = B(q, I).*

Condition 1 indicates that a birthmark is an information extracted by running p and is not additional information. In other words, birthmark does not require additional information likes software watermarks. Condition 2 means that the same birthmark can be obtained from a copied program. If birthmarks $B(p, I)$ and $B(q, I)$ are different, it means that q is not a copy of p. However, even if the same program, the birthmarks may vary depending on the inputs ($B(p, I) \neq B(p, J)$).

Two properties known as resilience and credibility should also ideally be satisfied.

Property 1: ***(Resilience)*** *For a p′ obtained by an arbitrary equivalent transformation of p, B(p, I) = B(p′, I) is satisfied.*

Property 2: ***(Credibility)*** *When programs p and q that develop independently, (B (p, I) ≠ B (p, I)) is satisfied.*

Resilience property indicates a resistance of birthmark to various types of attacks. Credibility property indicates that programs created completely independently can be differentiated even if their specifications are the same. The birthmarks that completely satisfy both these properties are difficult to create. Therefore, in practice, the strength of the birthmark must be set appropriately at the discretion of the user. Also, by Condition 1, birthmark information can be constructed without providing special information for the program.

The different kinds of dynamic birthmarks were proposed by various researchers. Each birthmark focuses on different runtime behaviors of the program. For example, the dynamic birthmarks based on execution path [3], API calls [4], runtime heaps [5], and etc. were proposed.

2.2. Similarities of the Birthmarks

The previous birthmark methods calculate similarities between two extracted birthmarks. The typical range of similarities is [0, 1] by the conventional papers. 0 means two programs are completely different, and 1 means the one program is strongly suspected a copy of the other. Then, the similarity between $B(p, I)$ and $B(q, I)$ is denoted by $\mathrm{sim}(B(p, I), B(q, I))$. Also, the threshold ε is introduced to decide the copy or not. We classify the score from sim into three groups to clarify the result from similarity, shows in the following equation [6].

$$\mathrm{sim}(B(p,I),B(q,I)) = \begin{cases} \geq \varepsilon & copy\ relation \\ \leq 1-\varepsilon & no\ copy\ relation \\ otherwise & inconclusive \end{cases}$$

If the similarity value is greater than ε, p and q have a copy relation, and if the similarity value is less than $1 - \varepsilon$, p and q are not a copy relation. In other cases, the birthmark method cannot conclude relation of two programs. The typical value of ε is 0.75 in the conventional papers [7].

3. THE PROPOSED METHOD

3.1. Motivation

Figure 1 depicts the typical scenario of the dynamic birthmarks. At first, we prepare two programs for comparing by the dynamic

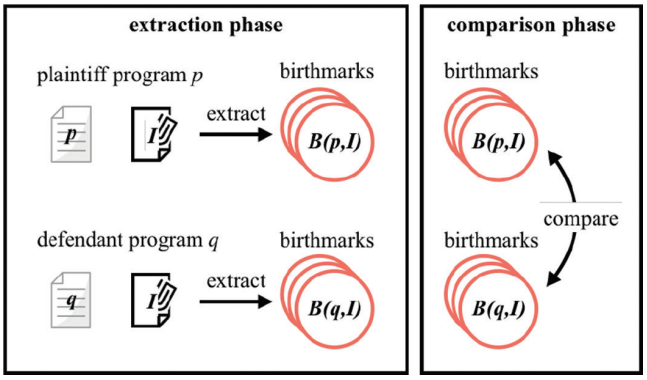


Figure 1 | Typical scenario of the dynamic birthmarks

birthmarks, and the inputs for them. One program is a plaintiff program, and the other is a defendant program. The inputs are quite important since the different input generates different birthmarks. Next, the dynamic birthmarks are extracted from two programs by running them with the prepared inputs (extraction phase). Finally, two extracted dynamic birthmarks are compared, and calculated similarity (comparison phase). The above procedures are called the birthmarking process.

The scenario is shared as the implied knowledge of the conventional papers. However, the conventional scenario involves the important issue. The issue is the time for the performing the scenario. The background of the issue is that the birthmark methods are to detect the program theft, not to prove. To detect the theft should examine the huge number of targets. Therefore, it is a quite serious issue that time is consumed for extracting a birthmark and comparing a pair of birthmarks. Because many birthmarks are extracted and the many pairs of birthmarks should be compared. For this, the goal of this paper is to reduce the total time of the birthmarking in focusing on the extraction phase.

The static birthmarks can be extracted easily since it requires only program itself. Therefore, reducing total time methods focus on comparing phase [8–10]. On the other hand, the dynamic birthmarks require the inputs for extracting them, and the extraction phase is conducted just before comparison. Therefore, automatically conducting the extraction phase contributes to reducing the total time. The time-consuming steps in the extracting phase are as follows. The one is to prepare the inputs for the programs since the inputs are prepared by the developers by fitting to the programs. The other one is to extract the dynamic birthmarks since to run the program consumes the time. For automatic extraction, this paper focuses on unit test codes on the projects as suitable inputs.

3.2. Extracting the Dynamic Birthmarks using the Unit Tests

The unit tests aim to find the bugs by running the target programs. The unit test codes generally give the concrete inputs to the methods of the target programs and examine the return values. Ideally, all methods of the target programs are tested by executing the unit test codes. Therefore, we can use the unit test codes as suitable inputs for extracting dynamic birthmarks.

On the other hand, the coverage is a metric for sufficiency evaluation of the unit tests, should be high as possible. The metric can

be used as the indicators of the inputs' diversities for the dynamic birthmark extraction.

In the conventional scenario, an evaluation by the birthmarks is conducted among a few plaintiff programs and a lot of defendant programs. Because the software theft is happened in anywhere and identity guaranteed the programs are hard to find. However, Open Source Software (OSS) is spread over the world, today. We can use OSS as the plaintiff programs. Therefore, the scenario in the paper is that we conduct the evaluation among a lot of plaintiff programs and a few defendant programs.

Figure 2 shows the overview of the proposed method. In the proposed method, we use OSS projects as a lot of plaintiff projects. Then, the dynamic birthmarks are extracted beforehand and store them into some databases. In the comparing phase, extracting the dynamic birthmarks pays the effort only from defendant programs. Finally, we compare between the dynamic birthmarks reading from databases and extracted above.

3.3. Automatic Extraction of the Dynamic Birthmarks

Aspect-Oriented Programming (AOP) is a programming paradigm that embeds the program code fragments [11]. The fragments are called cross-cutting concern and usually compose of common code (e.g. logging). Recently, AOP is used as the implementation of Dependency Injection (DI) [12].

The target of the paper is the projects. A project has a set of target programs and a set of unit test codes. The target programs are the product codes of the project. Therefore, we extract the dynamic birthmarks from the target programs by weaving the extracting codes with AOP and execute unit tests. Then, the dynamic birthmarks are extracted through the weaved codes.

Now, the below of the section describes the notation of the proposed method. At first, let R be a project, $P = \{p_1, p_2,..., p_n\}$ be a set of product codes, and $T = \{t_1, t_2,..., t_m\}$ be a set of test codes. Also, let $w_{\mathcal{B}}$ be an aspect code to extract a certain type of dynamic birthmark. Note that, a program of the unit tests has several test methods, thus, let each method be a test code. Then, we weave $w_{\mathcal{B}}$ into P, and obtain a set of weaved product codes $A_k = \{a_1, a_2,..., a_n\}$. The inputs for P is in each test code t_i, therefore, we regard t_i as an input ($1 \le i \le m$). Therefore, we denote $\mathcal{B}(P,T) = \{B_1(P,t_1), B_2(P,t_2),..., B_m(P,t_m)\}$ is the dynamic birthmarks for the proposed method.

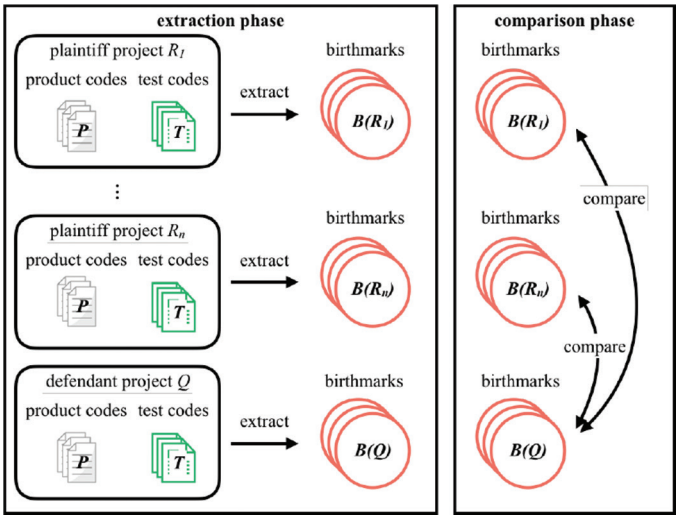


Figure 2 | The overview of the proposed method

3.4. Comparing the Dynamic Birthmarks

The different input generally generates the different birthmark, as mentioned in Section 3.1. However, it is hard to unify the inputs for the different projects. Because the format of inputs generally differs. Therefore, this paper does not unify the inputs for the projects, uses the unit test codes as the raw inputs. Moreover, the proposed method extracts several birthmarks, since a project has several test codes.

In other words, the two projects $R_x = \{P_x, T_x\}$ and $R_y = \{P_y, T_y\}$ are given, the birthmarks are extracted as $\mathcal{B}_x = \{B_{x,1}(P_x,t_{x,1}),..., B_{x,m_x}(P_x,t_{x,m_x})\}$, and $\mathcal{B}_y = \{B_{y,1}(P_y,t_{y,1}),..., B_{y,m_y}(P_y,t_{y,m_y})\}$. Then, $m_x \times m_y$ matrix is generated by comparing between every pair of birthmarks. Besides, each birthmark in $\mathcal{B}$ is denoted as $b_{i,j} = B_{i,j}(P_i, t_{i,j})$.

$$m(\mathcal{B}_x,\mathcal{B}_y) = \begin{pmatrix} \mathrm{sim}(b_{x,1},b_{y,1}) & \cdots & \mathrm{sim}(b_{x,m_x},b_{y,1}) \\ \mathrm{sim}(b_{x,1},b_{y,2}) & \cdots & \mathrm{sim}(b_{x,m_x},b_{y,2}) \\ \vdots & \ddots & \vdots \\ \mathrm{sim}(b_{x,1},b_{y,m_y}) & \cdots & \mathrm{sim}(b_{x,m_x},b_{y,m_y}) \end{pmatrix}$$

Then, we consider $m(\mathcal{B}_x, \mathcal{B}_y)$ to the cost matrix, and apply maximum weighted bipartite matching algorithms [13]. Next, we find pairs in order to be the maximum value among all possible matchings. Finally, the similarity between $\mathcal{B}_x$ and $\mathcal{B}_y$ are the average value of the pairs [14].

4. EXPERIMENTAL EVALUATION

4.1. The Target Project in the Experiment

This paper evaluates the proposed method focuses on the extraction cost and the credibility and resilience performances shown in the definition. For this, we compare the dynamic birthmarks extracted from OSS projects. The evaluation exploits three categories, three projects, and four version products (3 categories × 3 projects × 4 versions = 36 products). Besides, the categories are libraries for command lines, JSON, and CSV. Each category and project are chosen by the number of stars. Note that, in this paper, a product means a corresponding executable, and its product source and unit test codes. The products are distinguished by its name and version. Also, a project is a set of products in the same name and different versions.

Table 1 shows the target projects in the experiments and Table 2 shows their metrics. The columns of Table 2 are that Version, $|P|$, $|T|$, $C0$, $C1$, and Release shows product versions, the number of product code files, number of test methods, statement coverage ($C0$), branch coverage ($C1$), and release year, respectively. The coverages of almost product increased by updating versions.

Table 1 | The selected target projects and their web pages

	Projects Web page
CLI	Commons CLI https://commons.apache.org/cli/ Args4j http://args4j.kohsuke.org/ JLine2 https://jline.github.io/jline2/
JSON	Gson https://github.com/google/gson Jettison https://github.com/jettison-json/jettison Svenson https://github.com/fforw/svenson
CSV	Esperio CSV http://www.espertech.com/esper Jdbi http://jdbi.org/ Super CSV https://super-csv.github.io/super-csv/

Table 2 | The number of product codes and test methods of products

Projects	Version	$\|P\|$	$\|T\|$	$C0$ (%)	$C1$ (%)	Release
Commons CLI	1.1	20	214	83	80	2007
	1.2	20	187	96	91	2010
	1.3	22	364	96	93	2015
	1.4	22	372	96	93	2017
Args4j	2.0.8	24	33	67	60	2008
	2.0.16	40	59	73	67	2009
	2.0.31	62	151	76	73	2014
	2.33	63	162	77	74	2015
JLine2	2.0	29	25	52	39	2009
	2.6	30	43	62	46	2012
	2.13	48	141	79	55	2015
	2.14.5	48	143	79	56	2017
Gson	1.1	74	204	68	60	2008
	2.4	62	966	84	79	2015
	2.8.0	63	1,014	83	79	2016
	2.8.2	63	1,014	83	79	2017
Jettison	1.0	27	49	49	42	2008
	1.3.1	37	80	48	40	2011
	1.3.7	38	115	54	47	2014
	1.3.8	38	119	54	47	2016
Svenson	1.4.2	60	117	69	69	2012
	1.5.0	64	123	63	63	2015
	1.5.7	74	160	63	65	2017
	1.5.8	74	160	63	65	2017
Esperio CSV	5.2.0	26	81	75	72	2015
	5.4.0	26	81	75	72	2016
	6.1.0	26	81	75	72	2017
	7.0.0	26	81	75	72	2017
Jdbi	2.14	156	163	59	53	2011
	2.40	222	266	67	59	2012
	3.0.0	164	344	67	57	2017
	3.0.2	169	358	67	57	2018
Super CSV	2.0.0	69	377	100	100	2012
	2.2.0	82	450	100	100	2014
	2.4.0	84	516	99	100	2015
	2.4.1	88	549	98	99	2016

In the evaluation, we assume that the product in different version is stolen. Therefore, the same products in different version expect to have high similarities. Also, the similarities among different projects should be low.

4.2. Extracting the Dynamic Birthmarks with AOP

In the experiments, EXEFREQ dynamic birthmark was used, which represents frequencies of method calls [4]. In the definition of EXEFREQ, the system libraries are given. Therefore, we set the packages starts with java and javax in the standard API in Java[1] as the system libraries[2].

We developed the aspect code shown in Fig. 3 for AspectJ 1.8.10[3], and weaved it to the product codes of the target products. Then,

```
// The import statements are omitted.
@Aspect
public class ExeFreqExtractor {
  Map<Long, List<String>> birthmarks =
    new TreeMap<>();
  List<String> results = new ArrayList<>();
  public ExeFreqExtractor() {
    Runtime.getRuntime()
      .addShutdownHook(
        new Thread(() -> output()));
  }
  @Pointcut("within(ExeFreqExtractor)")
  public void runAspect() {   }
  @Pointcut("call(* java..*(..))")
  public void callAPI() {   }
  @Pointcut("execution(* * ..*test*(..))")
  public void execTest() {   }
  public List<String> initList(Long tId) {
    return birthmarks.getOrDefault(tId,
      new ArrayList<>());
  }
  @Before("callAPI()
    && !cflow(runAspect())")
  public void beforeWeave(JoinPoint jp) {
    addBirthmark(jp.toString());
  }
  @Before("execTest()
    && !cflow(runAspect())")
  public void beforeTest(JoinPoint jp) {
    if(!jp.toString().matches(".*junit.*"))
      results.add(jp.toString());
  }
  @After("execTest()
    && !cflow(runAspect())")
  public void afterTest(JoinPoint jp) {
    if(!jp.toString().matches(".*junit.*"))
      formatForOutput();
    birthmarks.clear();
  }
  private String mapToString(long id,
      List<String> list) {
    return String.format("%d %s",
      id, listToString(list));
  }
```

Figure 3 | The aspect code for extracting EXEFREQ birthmarks

[1] https://docs.oracle.com/javase/8/docs/api/
[2] The standard API in Java includes other packages (e.g., org.w3c.dom, and etc.), however, we omitted them for easy to understand in this paper.
[3] https://www.eclipse.org/aspectj/

```
  public void addBirthmark(String api) {
    Long tId = Thread
      .currentThread().getId();
    List<String> list = initList(tId);
    list.add(api);
    birthmarks.put(tId, list);
  }
  public void formatForOutput() {
    results.add(birthmarks.entrySet()
      .stream()
      .map(e -> mapToString(
          e.getKey(), e.getValue()))
      .collect(Collectors.joining(", ")));
  }
  private String
        listToString(List<String> list) {
    return toFreq(list).entrySet()
      .stream()
      .map(e -> String.format(
        "%s:%d", e.getKey(), e.getValue()))
      .collect(Collectors.joining(" "));
  }
  public Map<String, Integer>
        toFreq(List<String> list) {
    return list.stream()
      .collect(Collectors.toMap(
        k -> k, v -> 1,
        (v1, v2) -> v1 + v2,
        LinkedHashMap::new));
  }
  public void output() {
    results.stream()
      .forEach(System.out::println);
  }
}
```

Figure 3 | *Continued*

the test codes in the products are executed on JUnit 5[4] platform in order to extract the EXEFREQ dynamic birthmarks.

Figure 3 is the aspect code for extracting EXEFREQ. In the proposed method, the code of Fig. 3 is weaved into the product codes and runs the unit tests for extracting EXEFREQ dynamic birthmarks.

In the code of Fig. 3, at first, the shutdown hook of JVM is registered in the constructor to output the extracted birthmarks. Next, the names of the test method are stored at beforeTest method, and the resultant birthmarks are outputted at afterTest method with the name of the test method. Then, since EXEFREQ is the frequencies of the called methods, beforeWeave method stores the called method into List. Finally, in output method, the registered called methods are converted to the frequencies by toFreq method and outputted by formatting by formatForOutput method. Note that, Fig. 3 collects thread id since the EXEFREQ birthmarks are collected for each thread from the definition [4].

4.3. Evaluation of Performance for the Dynamic Birthmarks

This experiment evaluates the accuracy of the proposed method in the aspect of the two properties shown in Section 2.1. Tables 3–5 show the results of comparing among products by the proposed method. The first and the next column of each table represent project names and their versions. Also, the first and next row also show the project names and their versions. The following rows and columns represent the similarities between corresponding versions of projects. In Tables 3–5, the bold font shows the similarities are greater than 0.75 which is the typical value of ε. Moreover, the similarities of the grayed pairs were under 0.25 (1– ε) in the same project.

The similarities between the latest version and the next recent version of each project were greater than 0.8 and were higher than the other pairs. Additionally, the similarities in grayed cells were under 0.25, mean the pair were not similar, even if it paired in the same project.

Table 3 | The results of the libraries of command lines

Product	Version	Common CLI				Args4j				JLine2			
		1.1	1.2	1.3	1.4	2.0.8	2.0.16	2.0.31	2.33	2.0	2.6	2.13	2.14.5
Commons CLI	1.1	1.000											
	1.2	0.577	1.000										
	1.3	0.396	0.646	1.000									
	1.4	0.389	0.643	**0.976**	1.000								
Args4j	2.0.8	0.117	0.083	0.051	0.050	1.000							
	2.0.16	0.186	0.162	0.118	0.116	0.733	1.000						
	2.0.31	0.286	0.355	0.307	0.302	0.320	0.476	1.000					
	2.33	0.267	0.353	0.307	0.303	0.294	0.436	**0.929**	1.000				
JLine2	2.0	0.014	0.010	0.006	0.006	0.015	0.016	0.009	0.008	1.000			
	2.6	0.029	0.026	0.020	0.019	0.020	0.023	0.020	0.019	0.121	1.000		
	2.13	0.020	0.022	0.028	0.028	0.009	0.012	0.016	0.018	0.048	0.185	1.000	
	2.14.5	0.020	0.021	0.028	0.028	0.009	0.012	0.016	0.018	0.047	0.185	**0.987**	1.000

[4] https://junit.org/junit5

Table 4 | The results of JSON libraries

Product	Version	Gson				Jettison				Svenson			
		1.1	2.2	2.8.0	2.8.2	1.0	1.3.1	1.3.7	1.3.8	1.4.2	1.5.0	1.5.7	1.5.8
Gson	1.1	1.000											
	2.4	0.177	1.000										
	2.8.0	0.165	**0.900**	1.000									
	2.8.2	0.165	**0.889**	**0.984**	1.000								
Jettison	1.0	0.130	0.080	0.074	0.072	1.000							
	1.3.1	0.130	0.105	0.099	0.096	0.716	1.000						
	1.3.7	0.140	0.131	0.126	0.123	0.568	**0.766**	1.000					
	1.3.8	0.137	0.131	0.126	0.124	0.557	**0.753**	**0.983**	1.000				
Svenson	1.4.2	0.071	0.013	0.013	0.012	0.040	0.039	0.041	0.040	1.000			
	1.5.0	0.082	0.016	0.015	0.013	0.045	0.045	0.047	0.046	**0.918**	1.000		
	1.5.7	0.103	0.020	0.019	0.018	0.056	0.056	0.066	0.065	**0.768**	**0.825**	1.000	
	1.5.8	0.103	0.020	0.019	0.018	0.056	0.056	0.066	0.064	**0.770**	**0.825**	**0.990**	1.000

Table 5 | The results of CSV libraries

Product	Version	Esperio CSV				Jdbi				Super CSV			
		5.2.0	5.4.0	6.1.0	7.0.0	2.14	2.40	3.0.0	3.0.2	2.0.0	2.2.0	2.4.0	2.4.1
Esperio CSV	5.2.0	1.000											
	5.4.0	**0.999**	1.000										
	6.1.0	**0.999**	**0.999**	1.000									
	7.0.0	**0.999**	**0.999**	**0.999**	1.000								
Jdbi	2.14	0.036	0.035	0.035	0.036	1.000							
	2.4	0.022	0.022	0.022	0.022	0.445	1.000						
	3.0.0	0.002	0.002	0.002	0.002	0.037	0.043	1.000					
	3.0.2	0.002	0.002	0.002	0.002	0.036	0.042	**0.990**	1.000				
Super CSV	2.0.0	0.015	0.015	0.015	0.015	0.089	0.159	0.020	0.020	1.000			
	2.2.0	0.014	0.014	0.014	0.014	0.060	0.135	0.026	0.025	**0.847**	1.000		
	2.4.0	0.021	0.021	0.021	0.021	0.070	0.137	0.039	0.038	0.662	**0.793**	1.000	
	2.4.1	0.019	0.019	0.019	0.019	0.065	0.131	0.040	0.039	0.509	0.623	**0.823**	1.000

Then, we investigated the differences among products, shown in Table 6. The columns show project name, compared *from*, and *to*. The columns labeled P_c, P_u, P_a, P_d are the number of product code files with exact matches, updates, additions, and deletions, respectively. Note that, additions and deletions mean that the files exist only in *to* or *from*. The ratio column is the percentage of the exact match between versions, calculated by $\frac{P_c}{P_c + P_u + P_a + P_d}$. Besides, the investigation was performed by diff command[5].

The ratio columns between the latest and the next recent version are greater than 0.8 in all products. The results illustrate that the similarities from the recent versions by the proposed method were certainly high.

On the other hand, the similarities among different projects were generally low, the max value was 0.355, which between Args4j 2.0.31 and Commons CLI 1.2. The results mean that the proposed method can distinguish the independent projects.

[5] https://www.gnu.org/software/diffutils/

4.4. Evaluation of Cost for Extracting the Dynamic Birthmarks

The experiment evaluates the cost of extraction for the dynamic birthmarks by the proposed method. Figure 4 shows that the flowchart depicts differences between the proposed method and conventional method. The dynamic birthmarks are extracted from runtime behavior of the programs, therefore, extracting them must run the programs. For running the programs, it requires the inputs. Hence, we must understand the target programs for preparing the inputs in the conventional methods. Additionally, the dynamic birthmarks are sensitively changed by the given inputs. Because different inputs perform different control flow in the program. Extracting the characteristics of the whole programs requires the multiple inputs. Therefore, the cost to prepare the inputs is generally high.

On the other hand, understanding the programs does not require with the proposed method. Since the coverages by the test codes usually high in the mature projects, the proposed method can extract the characteristics of the whole programs. Therefore, the cost of extracting the dynamic birthmark with the proposed method are generally low than the conventional methods.

Table 6 | Similarities between product code in same project

	from	*to*	P_c	P_u	P_a	P_d	Ratio
Commons CLI	1.1	1.2	2	18	0	0	0.090
	1.1	1.3	0	20	2	0	0.000
	1.1	1.4	0	20	2	0	0.000
	1.2	1.3	0	20	2	0	0.000
	1.2	1.4	0	20	2	0	0.000
	1.3	1.4	18	4	0	0	0.818
Args4j	2.0.8	2.0.16	10	14	16	0	0.250
	2.0.8	2.0.31	1	23	38	0	0.016
	2.0.8	2.33	1	23	39	0	0.016
	2.0.16	2.0.31	7	32	23	1	0.111
	2.0.16	2.33	7	32	24	1	0.109
	2.0.31	2.33	51	11	1	0	0.810
JLine	2.0	2.6	1	25	4	3	0.030
	2.0	2.13	0	26	22	3	0.000
	2.0	2.14.5	0	26	22	3	0.000
	2.6	2.13	0	30	18	0	0.000
	2.6	2.14.5	0	30	18	0	0.000
	2.13	2.14.5	47	1	0	0	0.979
Gson	1.1	2.4	0	23	39	51	0.000
	1.1	2.8.0	0	23	40	51	0.000
	1.1	2.8.2	0	23	40	51	0.000
	2.4	2.8.0	25	37	1	0	0.397
	2.4	2.8.2	22	40	1	0	0.349
	2.8.0	2.8.2	51	12	0	0	0.810
Jettison	1.0	1.3.1	9	17	11	1	0.237
	1.0	1.3.7	5	21	12	1	0.128
	1.0	1.3.8	5	21	12	1	0.128
	1.3.1	1.3.7	18	19	1	0	0.474
	1.3.1	1.3.8	18	19	1	0	0.474
	1.3.7	1.3.8	31	7	0	0	0.816
Svenson	1.4.2	1.5.0	48	12	4	0	0.750
	1.4.2	1.5.7	30	30	14	0	0.405
	1.4.2	1.5.8	30	30	14	0	0.405
	1.5.0	1.5.7	35	29	10	0	0.473
	1.5.0	1.5.8	35	29	10	0	0.473
	1.5.7	1.5.8	73	1	0	0	0.986
Esperio CSV	5.2.0	5.4.0	25	1	0	0	0.962
	5.2.0	6.1.0	0	26	0	0	0.000
	5.2.0	7.0.0	0	26	0	0	0.000
	5.4.0	6.1.0	0	26	0	0	0.000
	5.4.0	7.0.0	0	26	0	0	0.000
	6.1.0	7.0.0	24	2	0	0	0.923
Jdbi	2.14	2.4	73	67	82	16	0.307
	2.14	3.0.0	0	39	125	117	0.000
	2.14	3.0.2	0	39	130	117	0.000
	2.4	3.0.0	0	47	117	175	0.000
	2.4	3.0.2	0	47	122	175	0.000
	3.0.0	3.0.2	154	10	5	0	0.911
Super CSV	2.0.0	2.2.0	0	69	13	0	0.000
	2.0.0	2.4.0	0	69	15	0	0.000
	2.0.0	2.4.1	0	69	19	0	0.000
	2.2.0	2.4.0	72	10	2	0	0.857
	2.2.0	2.4.1	69	13	6	0	0.784
	2.4.0	2.4.1	78	6	4	0	0.886

5. CONCLUSION

This paper aims to extract the dynamic birthmarks beforehand for reducing the total time of the birthmarking process. For this, the proposed method focuses on the unit tests in a project, and OSS projects as plaintiff programs. We weave an aspect code for extracting the target birthmarks to the product codes, then the birthmarks are extracted by running the unit tests.

In the experiments, the two properties of the birthmarks were evaluated, credibility and resilience performance. In the same project, the similarities between most recent two versions were

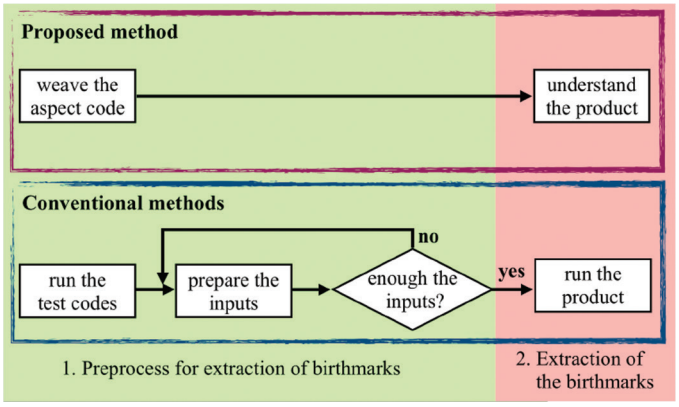


Figure 4 | The flowchart of the proposed method and the conventional method

greater than 0.8. On the other hand, the similarities among different projects were generally low, less than 0.355. Additionally, we compared the cost of extracting the birthmarks between the conventional method and the proposed method, the proposed method can reduce the extraction cost.

In our further works, we will conduct scaling up of experiments, implement extraction aspect code for other types of birthmarks, and evaluate the quality of birthmarks extracted by the proposed method.

ACKNOWLEDGMENT

Part of this work was supported by JSPS KAKENHI Grant Numbers 17K00196, 17K00500, and 17H00731.

REFERENCES

[1] H. Tamada, M. Nakamura, A. Monden, K. Matsumoto, Design and evaluation of birthmarks for detecting theft of java programs, in: Proceedings of the IASTED International Conference on Software Engineering (IASTED SE 2004), Innsbruck, Austria, February 2004, pp. 569–575.

[2] H. Tamada, M. Nakamura, A. Monden, K. Matsumoto, Java birthmarks — detecting the software theft —, IEICE Trans. Inf. Syst. E88-D (2005), 2148–2158.

[3] G. Myles, C.S. Collberg, Detecting software theft via whole program path birthmarks, in: Proceedings of the Information Security the 7th International Conference (ISC 2004), Springer-Verlag, Berlin Heidelberg, 27–29 September 2004, pp. 404–415.

[4] H. Tamada, K. Okamoto, M. Nakamura, A. Monden, K-i. Matsumoto, Dynamic Software Birthmarks to Detect the Theft of Windows Applications, in: Proceedings of the International Symposium on Future Software Technology 2004 (ISFST 2004), October 2004, pp. CD-ROM.

[5] P.P.F. Chan, L.C.K. Hui, S.M. Yiu, JSBiRTH: dynamic JavaScript birthmark based on the run-time heap, in: Proceedings of the 35th Annual Computer Software and Applications Conference (COMPSAC), IEEE, Munich, Germany, 18-22 July 2011, pp. 407–412.

[6] Z. Tian, Q. Zheng, T. Liu, M. Fan, DKISB: dynamic key instruction sequence birthmark for software plagiarism detection, in: Proceedings of the 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), IEEE, Zhangjiajie, 13-15 November 2013, pp. 619–627.

[7] D. Schuler, V. Dallmeier, C. Lindig, A dynamic birthmark for Java, in: Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), ACM, New York, NY, USA, 5-9 November 2007, pp. 274–283.

[8] J. Nakamura, H. Tamada, Fast comparison of software birthmarks for detecting the theft with the search engine, in: Proceedings of the 4th International Conference on Applied Computing & Information Technology (ACIT 2016), IEEE, Las Vegas, NV, USA, December 2016.

[9] J. Nakamura, H. Tamada, mituba: scaling up software theft detection with the search engine, in: Proceedings of the International Conference on Software Engineering and Information Management (ICSIM 2018), Casablanca, Morocco, January 2018, pp. 6–10.

[10] T. Tsuzaki, T. Yamamoto, H. Tamada, A. Monden, Scaling up software birthmarks using fuzzy hashing, Int. J. Software Innov. (IJSI), 5 (2017), 89–102.

[11] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, J. Irwin, Aspect-oriented programming, in: Proceedings of the European Conference on Object-Oriented Programming (ECOOP 1997), Springer-Verlag, Finland, June 1997, pp. 220–242.

[12] D.R. Prasanna, Dependency Injection. Manning Publications Co., 1st ed., New York, USA, 2009.

[13] H.W. Kuhn, On the origin of the Hungarian method, in: J.K. Lenstra, A.H.G. Rinnooy Kan, A. Schrijver (Eds.), History of Mathematical Programming, CWI, Amsterdam and North-Holland, Amsterdam, 1991, pp. 77–81.

[14] Z. Tian, T. Liu, Q. Zheng, E. Zhuang, M. Fan, Z. Yang, Reviving sequential program birthmarking for multithreaded software plagiarism detection, IEEE Transactions on Software Engineering, 44 (2017), 491–511.