Software Bertillonage

Finding the Provenance of Software Entities

Mike Godfrey
Software Architecture Group (SWAG)
University of Waterloo

Work with …

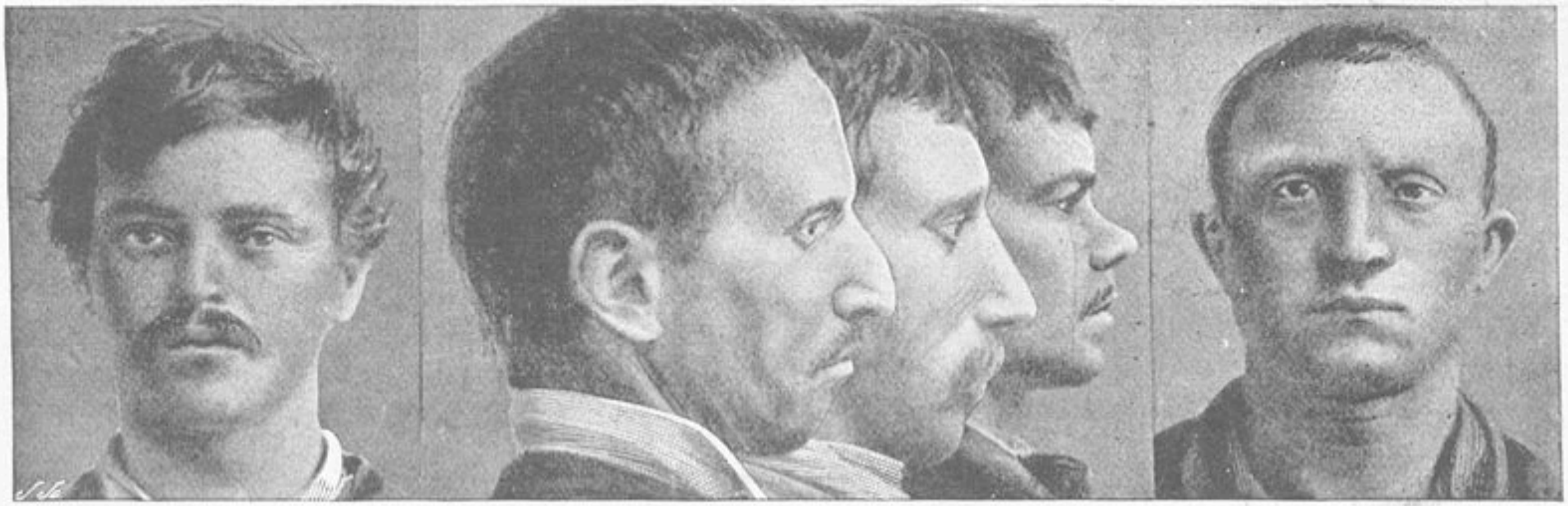
- Daniel German
- Julius Davies**
- Abram Hindle

- Wei Wang
- Ian Davis
- Cory Kapser
- Lijie Zou
- Qiang Tu

** Did most of the hard work

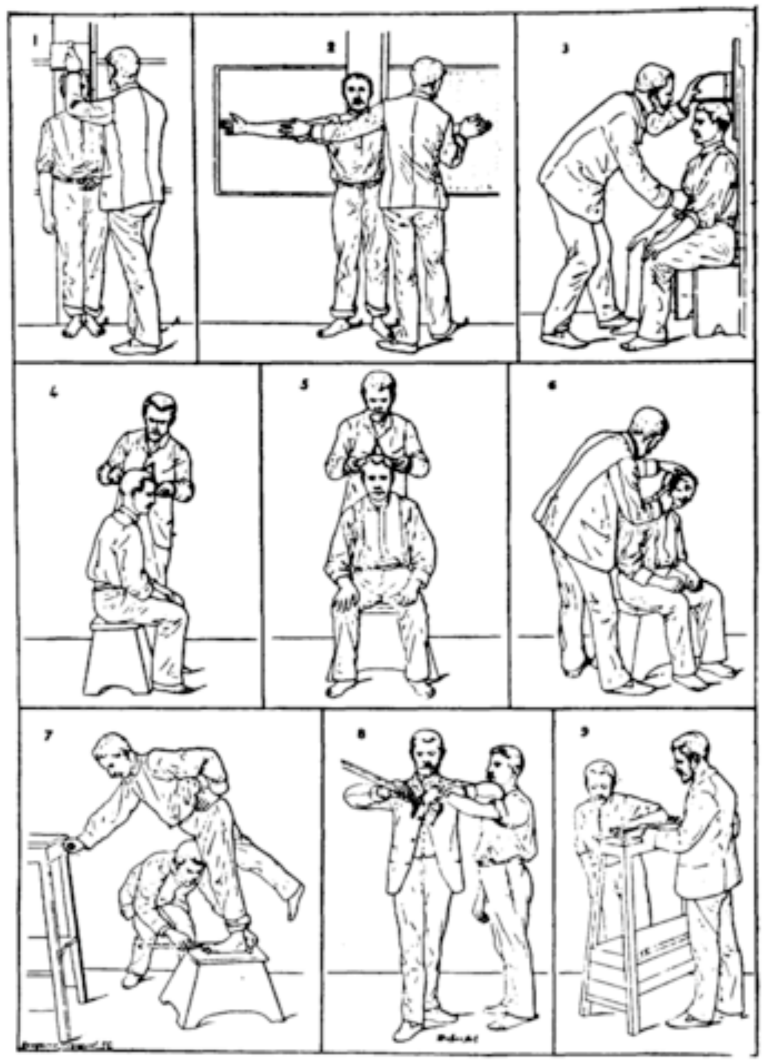
Who are you?

Alphonse Bertillon
(1853-1914)



The nose, as it cannot be disguised, is extremely important in identification. The types above, taking them from the left, show a low, narrow nose, a hooked nose, a straight nose, a snub nose, and a high, wide nose.

RELEVÉ
DU
SIGNALEMENT ANTHROPOMÉTRIQUE



1. Taille. — 2. Envergure. — 3. Buste. —
4. Longueur de la tête. — 5. Largeur de la tête. — 6. Oreille droite. —
7. Pied gauche. — 8. Médius gauche. — 9. Coudée gauche.

Bertillonage metrics

1. Height
2. Stretch: Length of body from left shoulder to right middle finger when arm is raised
3. Bust: Length of torso from head to seat, taken when seated
4. Length of head: Crown to forehead
5. Width of head: Temple to temple
6. Length of right ear
7. Length of left foot
8. Length of left middle finger
9. Length of left cubit: Elbow to tip of middle finger
10. Width of cheeks

Forensic Bertillonage

- Quick and dirty, and a huge leap forward
  - Some training and tools required but could be performed with technology of late 1800s
  - If done accurately, could quickly narrow down a very large pool of mugshots to only a handful

- Problems:
  - Equipment was cumbersome, expensive, required training
  - Measurement error, consistency
  - The metrics were not independent!
  - Adoption (and later abandonment)

Software Bertillonage

- We want quick and dirty ways of looking at a function (file, library, binary, etc) and asking:
  - *Who are you, really?*
    - *Entity and relationship analysis*
  - *Where did you come from?*
    - *Evolutionary history*
  - *Does your mother know you're here?*
    - *Licensing*

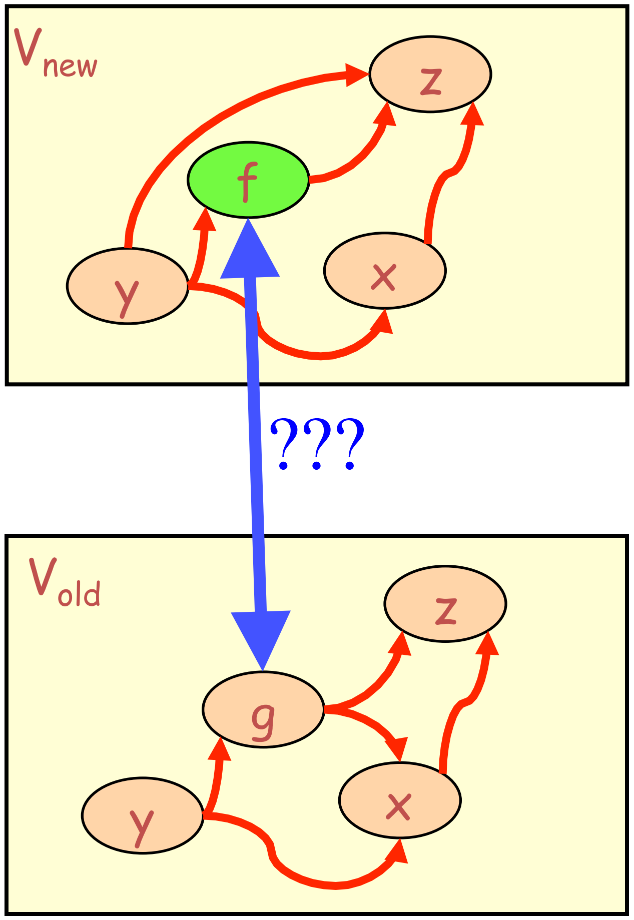
Related ideas

- Software clone detection
  - Why?
    - Just “understand” where/why duplication has occurred
    - Possible refactoring to reduce inconsistent maintenance, binary footprint size, to improve design, …
    - Tracking software licensing compatibilities, esp. included libraries and cross-product entity “adoption”
  - Many techniques for this ☺

Related ideas

- "Origin analysis" + sw genealogy
  - Why?
    - Program comprehension
    - Name / location change of sw entity within a system can break longitudinal studies
  - Use entity and relationship analysis to look for likely suspects

[IWPC-02, WCRE-03, TSE-05]



Related ideas

- MSR, bug predictors, and SE recommender systems
  - Why?
    - Given info about similar situations, what might be helpful / informative in this situation?
  - Many techniques (AI, LSI, LDA, data mining, plus ad hoc specializations + combinations)

- … and so on …

Bertillonage desiderata

- A good Bertillonage metric should:
  - be computationally inexpensive
  - be applicable to the desired level of granularity and programming language
  - catch most of the bad guys (recall)
  - significantly reduce the search space (precision)
- Bertillonage is not fingerprinting or DNA analysis!
  - Often there just is not enough info (or too much noise) to make conclusive identification
  - So we hope to reduce the candidate set so that manual examination is feasible

Bertillonage meta-techniques

1. Count based

   e.g., size, LOC, fan-in, McCabe

2. Set based

   e.g., contained string literals, method names

3. Relationship based

   e.g., call sets, throws sets, libraries included / used

4. Sequence based

   e.g., methods in order, tokens-based clone detection

5. Graph based

   e.g., AST and PDG clone detection

A problem

- Software packages often bundle in third-party libraries to avoid "DLL-hell" [Di Penta-10]
  - In Java world, jars may include library source code or just byte code
  - Included libs may include other libs too!

- Payment Card Industry Data Security Std (PCI-DSS), Req #6:
  - *"All critical systems must have the most recently released, appropriate software patches to protect against exploitation and compromise of cardholder data."*

What if a financial software package doesn't explicitly list the version IDs of its included libraries?

Identifying included libraries

- The version ID may be embedded in the name of the component!

  e.g., `commons-codec-1.1.jar`

  – … but often the version info is simply not there!

- Use fully qualified name of each class plus a code search engine [Di Penta 10]
  – Won't work if we don't have library source code

- Compare against all known compiled binaries
  – But compilers, build-time compilation options may differ

Anchored class signatures

- Idea: Compile / acquire all known lib versions but extract only the signatures, then compare against target binary
  - Shouldn't vary by compiler/build settings

- For a class C with methods $M_1, \ldots, M_n$, we define its *anchored class signature* as:

$$\theta(C) = \langle \sigma(C), \langle \sigma(M_1), \ldots, \sigma(M_n) \rangle \rangle$$

- For an archive A composed of classes $C_1,\ldots,C_k$, we define its *anchored class signature* as

$$\theta(A) = \{\theta(C_1), \ldots, \theta(C_k)\}$$

```
// This is **decompiled** source!!
package a.b;

public class C extends java.lang.Object
        implements g.h.I {

    public C() {
        // default constructor is inserted by javac
    }

    synchronized static int a (java.lang.String s)
            throws a.b.E {
        // decompiled byte code omitted
    }
}
```

$\sigma(C)$ = `public class a.b.C extends Object implements I`
$\sigma(M_1)$ = `public C()`
$\sigma(M_2)$ = `default synchronized static int a(String) throws E`

$\theta(C) = \langle \sigma(C), \langle \sigma(M_1), \sigma(M_2) \rangle\rangle$

Archive similarity

- We define the *similarity index* of two archives as their Jaccard coefficient:

$$sim(A,B) = \frac{|\theta(A) \cap \theta(B)|}{|\theta(A) \cup \theta(B)|}$$

- We define the *inclusion index* of two archives as:

$$inclusion(A,B) = \frac{|\theta(A) \cap \theta(B)|}{|\theta(A)|}$$

Implementation

- Created byte code (bcel5) and source code signature extractors
- Used SHA1 hash for class signatures to improve performance
  - We don't care about near misses at the method or class level!
- Built corpus from Maven2 jar repository
  - Maven is unversioned + volatile!
  - 150 GB of jars, zips, tarballs, etc.,
  - 130,000 binary jars (75,000 unique)
  - 26M .class files, 4M .java source files (incl. duplicates)
  - Archives contain archives: 75,000 classes are nested 4 levels deep!

Investigation

Target system: An industrial e-commerce application containing 84 jars.

RQ1: How useful is the archive signature similarity index at finding the original binary archive for a given binary archive?

RQ2: How useful is the archive signature similarity index at finding the original sources for a given binary archive?

RQ3: How reliable is the version information stored in a jar file's name?

Investigation

RQ1: How useful is the archive signature similarity index at finding the original binary archive for a given binary archive?

- 51 / 84 binary jars (60.7%), we found a single (correct) candidate from the corpus with similarity index of 1.0.
- 20 / 84 we found multiple matches with *simIndex* = 1.0
- 12 / 84 we found no matches with *simIndex* = 1.0
  - But 10 / 12 we found correct product
- 1 / 84 we found no match (product was not in Maven)

More data here: http://juliusdavies.ca/uvic/jarchive/

Summary

- Who are you?
  - Determining the provenance of software entities is a growing and important problem

- Software Bertillonage:
  - Quick and dirty techniques applied widely, then expensive techniques applied narrowly

- Identifying version IDs of included Java libraries is an example of the software provenance problem
  - And our solution is an example of software Bertillonage

Non-CS References

- *Fingerprints: The Origins of Crime Detection and the Murder Case that Launched Forensic Science*, Colin Beavan, Hyperion Publishing, 2001.
- http://en.wikipedia.org/wiki/Alphonse_Bertillon

Software Bertillonage

Finding the Provenance of Software Entities

Mike Godfrey

Software Architecture Group (SWAG)

University of Waterloo