CS846 course evals

https://perceptions.uwaterloo.ca/

(Formerly, evaluate.uwaterloo.ca)

- Responses are anonymous!
- I will not get access to them until next term, after your final grades have been submitted
- I'll be able see the *comments* you make (but no one else will)
- The *numeric scores* will be viewable by my chair (and me) and will be used to evaluate my performance as a professor

Also …

- Please send me a list of your favourite 3 and least favourite 3 papers from the term
  - I use this feedback to decide which one to use again
- If you don't, I will … ask you again politely. And again. And again.

Does your mother know you're here?

Understanding software artifact provenance

Mike Godfrey
Software Analytics Group
David R. Cheriton School of Computer Science
University of Waterloo

UNIVERSITY OF WATERLOO

Overview

- Two motivating problems
- Software artifact provenance
  - An emerging problem area in software analytics
- Software Bertillonage
  - A metaphor for attacking the provenance problem

A problem

- You're the project manager of a (closed source) web management system.
- You get an email message:

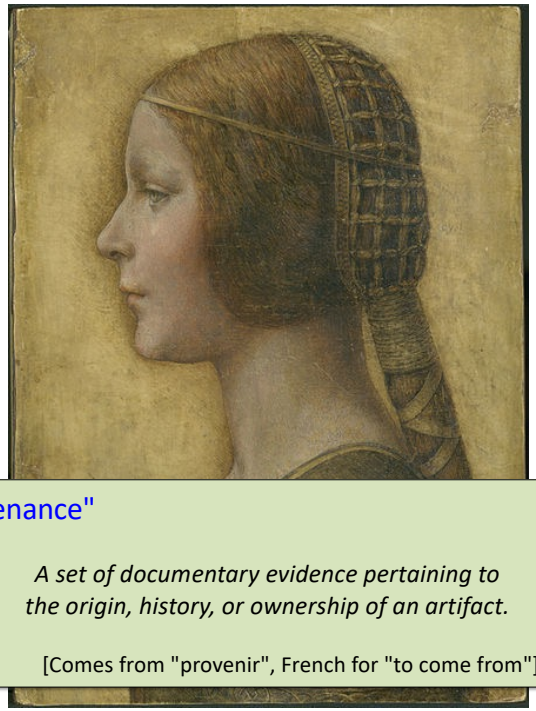
> Hi, I'm the project leader of an open source alternative to your product.
>
> I think someone on your team has stolen some of our GPL code to implement features X, Y, and Z in your product.
>
> Could you please investigate and respond ASAP.

Another problem

- Your company uses a commercial Java application that ships with embedded third-party libraries
  - One of these libraries had a major security flaw in version 2.1 that was fixed in version 2.1.1
  - Are you vulnerable? How can you tell?

Software artifact provenance

An emerging problem area



"Provenance"

A set of documentary evidence pertaining to the origin, history, or ownership of an artifact.

[Comes from "provenir", French for "to come from"]

Software artifact provenance

- For a given function, class, file, library, binary, bug report, feature, test suite, … we want to investigate its origin, evolution, and the supporting evidence to be able to answer:
  - *Who are you, really?*
  - *Where did you come from?*
  - *Are there any more like you at home?*
  - *Does your mother know you're here?*
- [This is not quite the same problem as software security provenance]

Software artifact provenance Example problems

- *Where was feature XXX implemented in the codebase?*
  - *How much did it cost to implement? maintain?*
  - *How much change has it undergone?*
  - *Where was it discussed in the developer mailing list?*
- *How much duplicated code is there in my system? Why?*
  - *How should I manage the duplication over time?*
  - *Does the cloning imply high-level design similarity?*
  - *Does the use of the duplicated code violate the GPL?*
  - *Does the latest release contain at least 25% "new code"?*

Software artifact provenance Example problems

- *Which version of library `httpclient.jar` is included in the default Android environment?*
- *Has anyone worked on a similar task before?*
  - *Is this bug report a duplicate?*
  - *How "similar" is this much smaller test suite to the original?*
  - *What APIs might be useful for this maintenance task? [Mylyn]*

… and what is the evidence?

Software artifact provenance

Investigating provenance

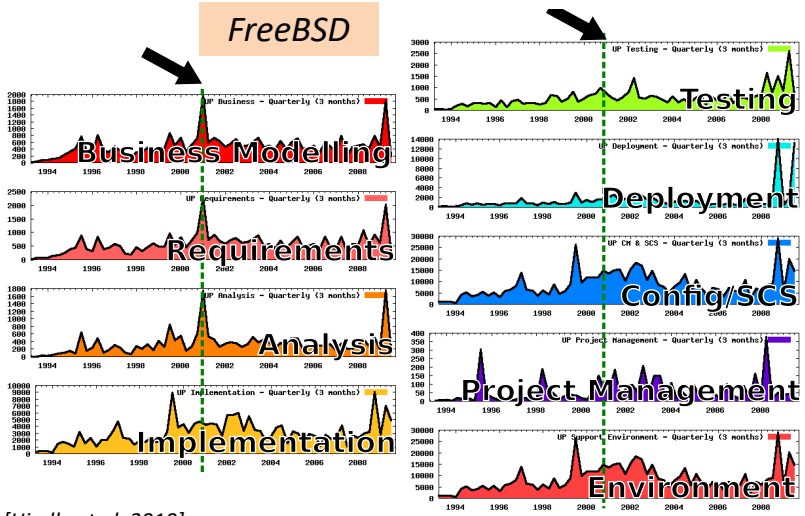
Two big tasks:

1. Scoping and identifying the entity of interest
   - *What's a feature? How big is a clone? What's a maintenance task?*
   - *What does "same" or "similar" mean? What thresholds are reasonable?*

2. Extracting and analyzing the evidence
   - Many kinds of evidence, analyses
   - Ground truth? Master repository? Missing data? S:N ratio?
   - Synthesis and analysis techniques must scale!

Software artifact provenance

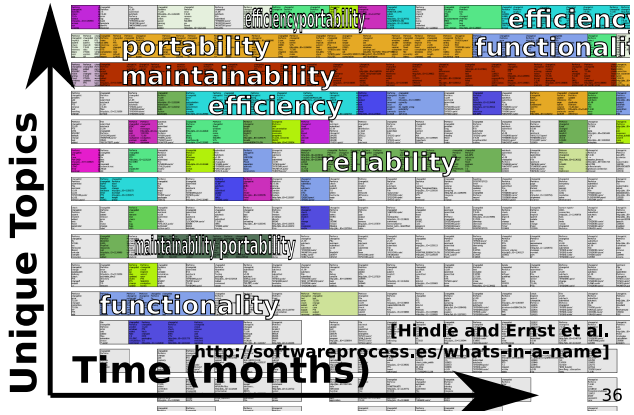
Software process extraction



[Hindle et al. 2010]

Software artifact provenance

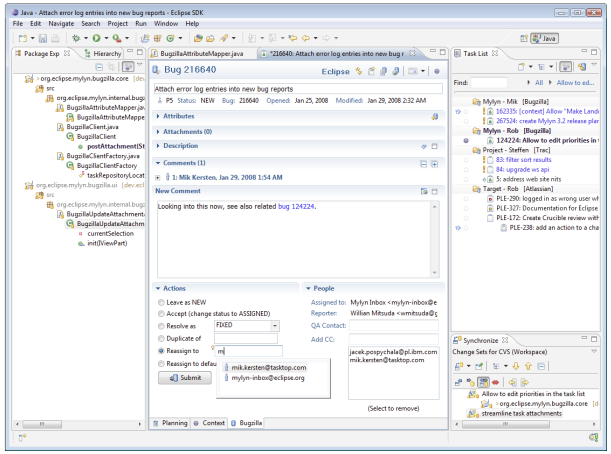
Commit message topic mining



[Hindle, Ernst, et al., MSR 2011 + *Empirical Sw Eng* 2013]

Software artifact provenance

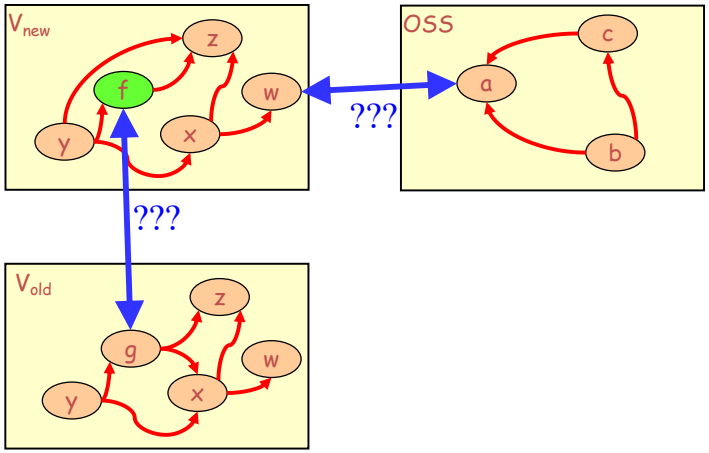
Recommender systems: Mylyn



[Kersten and Murphy, AOSD 2005]

Software artifact provenance

Origin analysis + copyright violation



[Godfrey and Zou, *IEEE Trans. on Sw Eng,* 2005]

Software artifact provenance

Source code cloning

```
// gnumeric
static PyObject *
py_new_RangeRef_object (const GnmRangeRef *range_ref){
   py_RangeRef_object *self;
   self = PyObject_NEW py_RangeRef_object,
       &py_RangeRef_object_type);
   if (self == NULL) {
    return NULL;
   }
   self->range_ref = *range_ref;
   return (PyObject *) self;
}
```

Software artifact provenance

Source code cloning

```
// gnumeric
static PyObject *
py_new_Range_object (GnmRange const *range) {
   py_Range_object *self;
   self = PyObject_NEW (py_Range_object,
       &py_Range_object_type);
   if (self == NULL) {
    return NULL;
   }
   self->range = *range;
   return (PyObject *) self;
}
```

Software artifact provenance

Source code cloning by intent

1. Forking
   - Hardware variation
     e.g., Linux SCSI drivers
   - Platform variation
   - Experimental variation
2. Templating
   - Boilerplating
   - API / library protocols
   - Generalized programming idioms
   - Parameterized code
3. Post-hoc customizing
   - Bug workarounds
   - Replicate + specialize

["'Cloning considered harmful' considered harmful: Patterns of cloning in software", Cory J. Kapser and Michael W. Godfrey, *Empirical Software Engineering,* 2008]

Software artifact provenance

Cloning harmfulness: Two case studies

		Apache		Gnumeric	
Group	**Pattern**	**Good**	**Harmful**	**Good**	**Harmful**
Forking	Hardware variation	0	0	0	0
Forking	Platform variation	10	0	0	0
Forking	Experimental variation	4	0	0	0
Templating	Boiler-plating	5	0	6	7
Templating	API	0	0	0	9
Templating	Idioms	0	12	1	1
Templating	Parameterized code	5	12	10	34
Customizing	Replicate + specialize	12	4	15	16
Customizing	Bug workarounds	0	0	0	0
Total		**36**	**28**	**32**	**67**

Apache httpd 2.2.4 - 60 Tokens
Gnumeric 1.6.3 - 60 Tokens

["'Cloning considered harmful' considered harmful: Patterns of cloning in software", Cory J. Kapser and Michael W. Godfrey, *Empirical Software Engineering*, 2008]

Software artifact provenance

The challenge

- Given the recent advances in Software Analytics …
  - … can we develop reliable techniques that take advantage of a myriad of inter-related artifact kinds to establish the provenance of a given software entity?
  - … and can we strategically minimize the amount of heavy analysis that we need to do?
- Some open problems:
  - Sensible, clear definition? Ground truth?
  - Artifact linkage? Dependable, tractable analyses?
  - Running matching algorithms on large datasets?

Software artifact provenance

Summary

- Given a software entity (*chunk of code, test suite, email topic)* :
  - *Who are you, really?*
  - *Where did you come from? Are there more like you at home?*
  - *Does your mother know you're here?*

Software Bertillonage

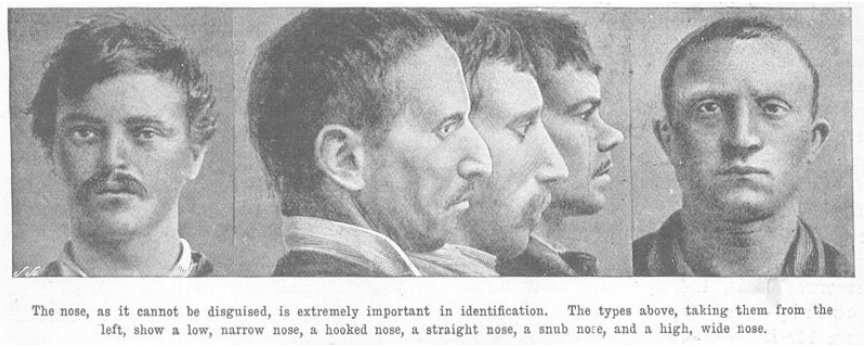
A metaphor for attacking the provenance problem

Who are you?

Alphonse Bertillon (1853-1914)

"[Holmes'] conversation, I remember, was about the Bertillon system of measurements, and he expressed his enthusiastic admiration of the French savant."

[A. Conan Doyle, *The Memoirs of Sherlock Holmes*]

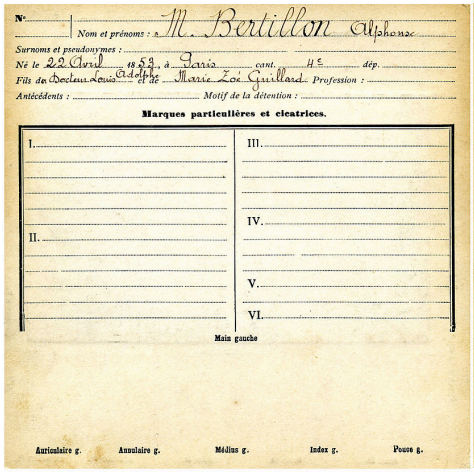


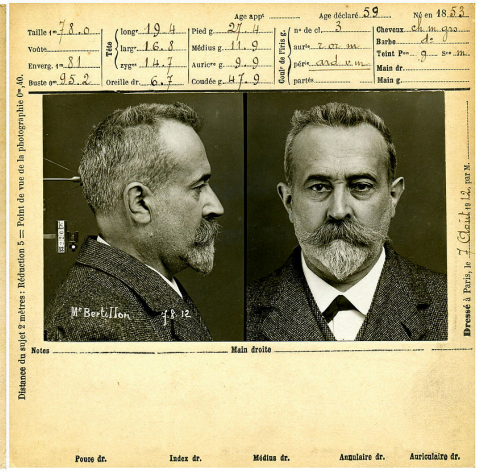
The nose, as it cannot be disguised, is extremely important in identification. The types above, taking them from the left, show a low, narrow nose, a hooked nose, a straight nose, a snub nose, and a high, wide nose.

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.

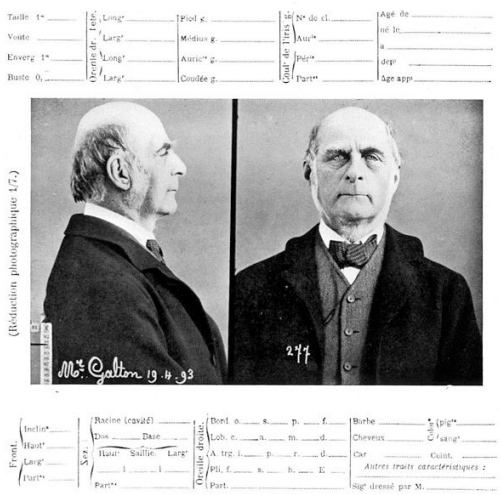
Forensic Bertillonage

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





Who are you?



"Software Bertillonage" (???)

- It's not fingerprinting or DNA analysis!
  - There may be not enough info / too much noise to make positive ID
  - You may be looking for a cousin or ancestor
  - You may be synthesizing something that doesn't exist elsewhere
- A good software Bertillonage metric should:
  - be computationally inexpensive
  - be applicable to the desired level of granularity / programming language
  - catch most of the bad guys (recall)
  - significantly reduce the search space (precision)

Software Bertillonage meta-techniques

1. Count/number based	*size, fan-in/out, McCabe, hash values*
2. Set based	*contained string literals, method names, sets of numbers*
3. Relationship based	*libraries included/used, calls/called-by, defines/uses, extends/implements, throws/thrown by*
4. Sequence based	*method invocation chains, token-based clone detection*
5. Graph based	*AST and PDG clone detection*

Software Bertillonage
Finding GPL violations via binary analysis

```
$ strings suspectSystem.exe      | sort > suspectStrings
$ strings gplSystem.exe          | sort > gplStrings
$ diff suspectStrings gplStrings | less
```

[Hemel, Kalleberg, Vermaas, + Dolstra, MSR 2012]

Software Bertillonage

A real-world problem

- Software packages often bundle in third-party libraries to avoid "DLL-hell" [Di Penta-10]
  - Java jars: [byte code [+ source]]*
- Payment Card Industry Data Security Std, 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?

Software Bertillonage

Matching anchored signatures

Q: *Which version of library* `httpclient.jar` *is included in this Java application?*

Our approach:

- Consider only class / method signatures
  - May not have source, compiler + options may differ, …
- Build master repos of SHA1 signature hashes for each class in Maven2
  - Which has gaps, duplication, errors, …
- Compare sig. hashes of target application against master repos
  - There will be false positives when API does not evolve
  - … so the effectiveness of narrowing search space depends on how much APIs evolve

[Davies, German, Godfrey, Hindle, MSR 2011 + *Empirical Sw Eng*, 2013]

Testing the extractors, sampling the data

- Julius built a hash corpus from Maven2 jar repository
  - Maven is unversioned + volatile!
- First attempt, Maven2 looked like this:
  - 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!

Anchored class signatures

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

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

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

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

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

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

Testing the extractors, sampling the data

- Randomly picked 1000 binary jars for which there was also a source jar in Maven2
  - # of classes per binary archive: median: 5, max: 2138
- Binary-to-binary matching (bin2bin):
  - Each binary archive matched itself ☺
  - # of exact matches in Maven (due to duplication or unchanging API)
    - median: 5, max: 487
- Binary-to-source matching (bin2src):
  - Correct match was among top matches (median:4, max: 158): 966 times
  - Something else was a better match (test classes): 30 times
  - No matches suggested (compiler/extractor issues): 4 times

Industrial case study

Target system: An industrial e-commerce app containing 84 jars

Q: How useful is the signature similarity index in finding the original *binary* archive for a given binary archive?

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

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

Further uses

1. Licensing and security audit of target e-commerce app:
   - One jar changed open source licenses (GNU Affero, LGPL)
   - One jar version was found to have known security bugs
2. *"When did Google Android developers copy-paste* `httpclient.jar` *classes into* `android.jar`*?"*
   - Very quickly, Julius narrowed it down to two likely candidates, one of which turned out to be correct

Summary

Software artifact provenance

Where did this thing come from?

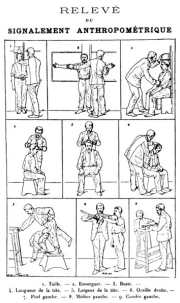
How do we know?

What else is like it?

Is its current use permissible?

Software Bertillonage

Cheap approx. techniques applied widely, then expensive precise techniques applied narrowly



Questions

- Who has experience in IP infringement?
  - What is the standard of evidence?
- Who has experience packaging third-party libraries for use with "your" application?
- How do you manage copy-paste programming?
- Do you have knowledge of software analytics enabling "bad management" practices?

Does your mother know you're here?

Understanding software artifact provenance

Mike Godfrey
Software Analytics Group
David R. Cheriton School of Computer Science
University of Waterloo

UNIVERSITY OF WATERLOO

SWAG