Towards the census of public source code history

Audris Mockus

[2009-05-30]

Premise: Code as Functional Knowledge

- Functional knowledge
  - Scholarly and literary works need a subject to interpret/perform them
  - Code just needs a computer to be executed
- Open source code
  - A vehicle for innovation through reuse (build on existing knowledge)
  - A common platform for everyone to express themselves (contribute

  their knowledge)
- Codebase for legacy systems encodes thousands of person-years of knowledge on:
  - the organization (process of producing the code) and
  - the market (value the software provides to users)

Implications

- Developers are `transient`, but the code is `everlasting`
- Developers can `only` leave a lasting impact
  - through `changes` to the code and
  - through `training` developers who succeed them
- Traces developers leave in the code shed light on how software is created and how developers interact
- Authorship as social signaling
- Code reuse is good, e.g., {"modules reused without revision had the fewest faults, fewest faults per source line, and lowest fault correction effort"} [2]
- What happens in a succession: when develpers change, but code stays?}

Why study global properties of code?

- How much code? What is that code? How old, of what type, where?
  - Extent of code transfer/reuse: study patterns or reuse and innovation
    - Full sample needed to avoid missing instances of reuse
  - Authorship (succession): Find Adam&Eve of code or identify original authors
    - Full sample needed to avoid missing first creators
  - License compliance: verify that code is not borrowed from public domain
    - Full sample needed to avoid missing instances of borrowing

Example uses: licensing and security

- License compliance
  - Check each version of each file in a firm to see if it matches or is similar to any of the public source code
  - Comparable services from Palamida or BlackDuck are very expensive
    - Track serious defects (e.g., security problems) and identify all other code (inside and outside Avaya) and products that have that code

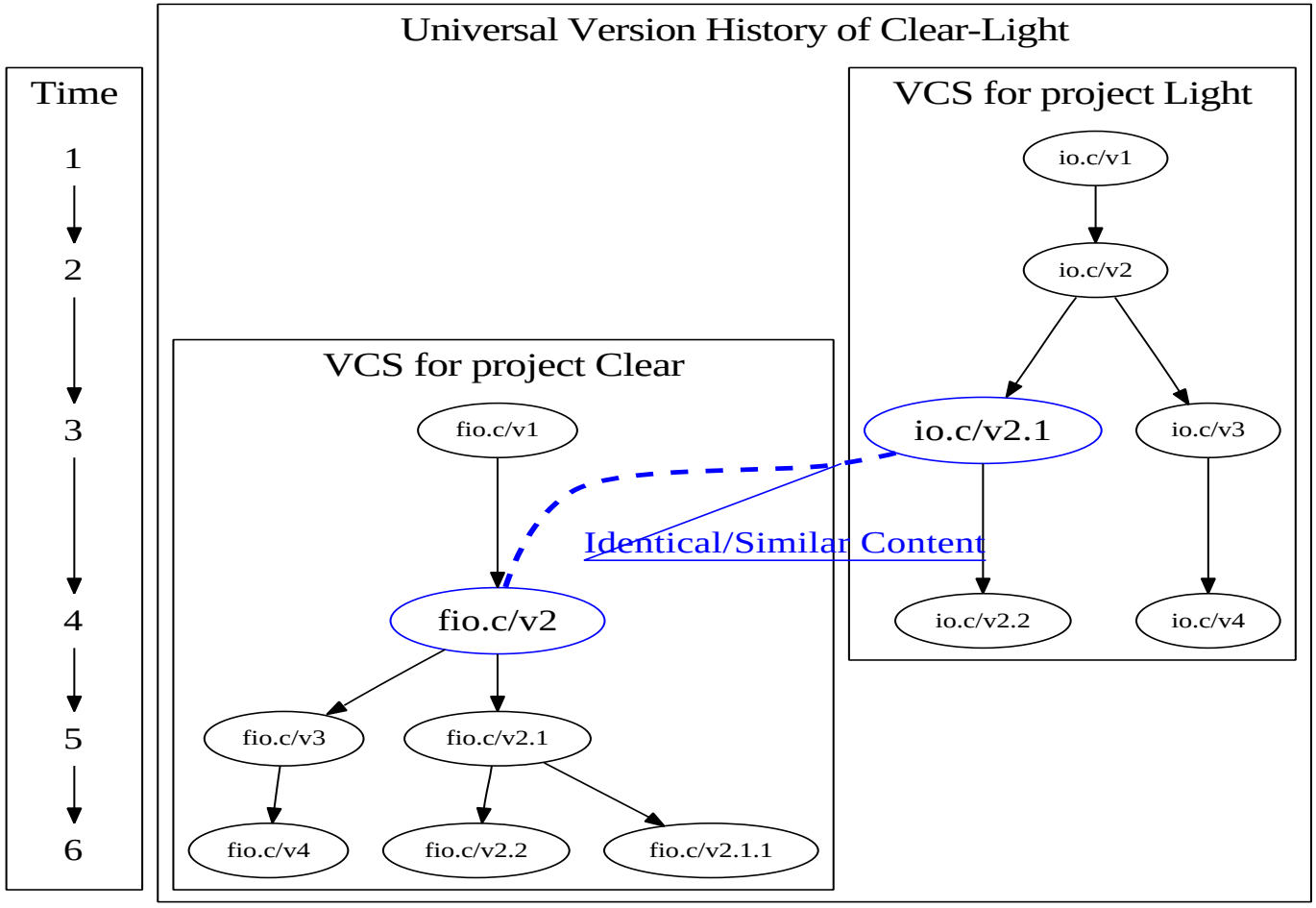
Approach: Version Control Census

- Gather all public VCSs
- Index all source code
- Use/Analyze this repository, e.g.,
  - Establish links across project repositories to create Universal Version History
  - Study (provide tools for) innovation, error propagation, authorship

How to conduct VCS Census?

- Discover VCS repositories
- Copy/clone repositories
- Extract and index all versions of each file
- Establish similarity among files to determine identity of each file
  - Unlike people, files and their version histories can be and very often are copied
  - To avoid double-count for census and other analysis we thus need to create each file's "passport" or provenance
- Conduct further analysis

Identity/provenance of the code



Discovery strategy

- Sites with many projects: e.g., SourceForge, GoogleCode, Savannah, repo.or.cz, github.com
- Ecosystems: e.g., Gnome, KDE, NetBeans, Mozilla, ...
- Famous: e.g., Mysql, Perl, Wine, Postgres, and gcc
- In wide use: e.g., git.debian.org
- Directories: e.g., RawMeat and FSF
- Published surveys of projects
- Verify search for common filenames on Google Code Search to see if new files are discovered

How to automate VCS discovery?

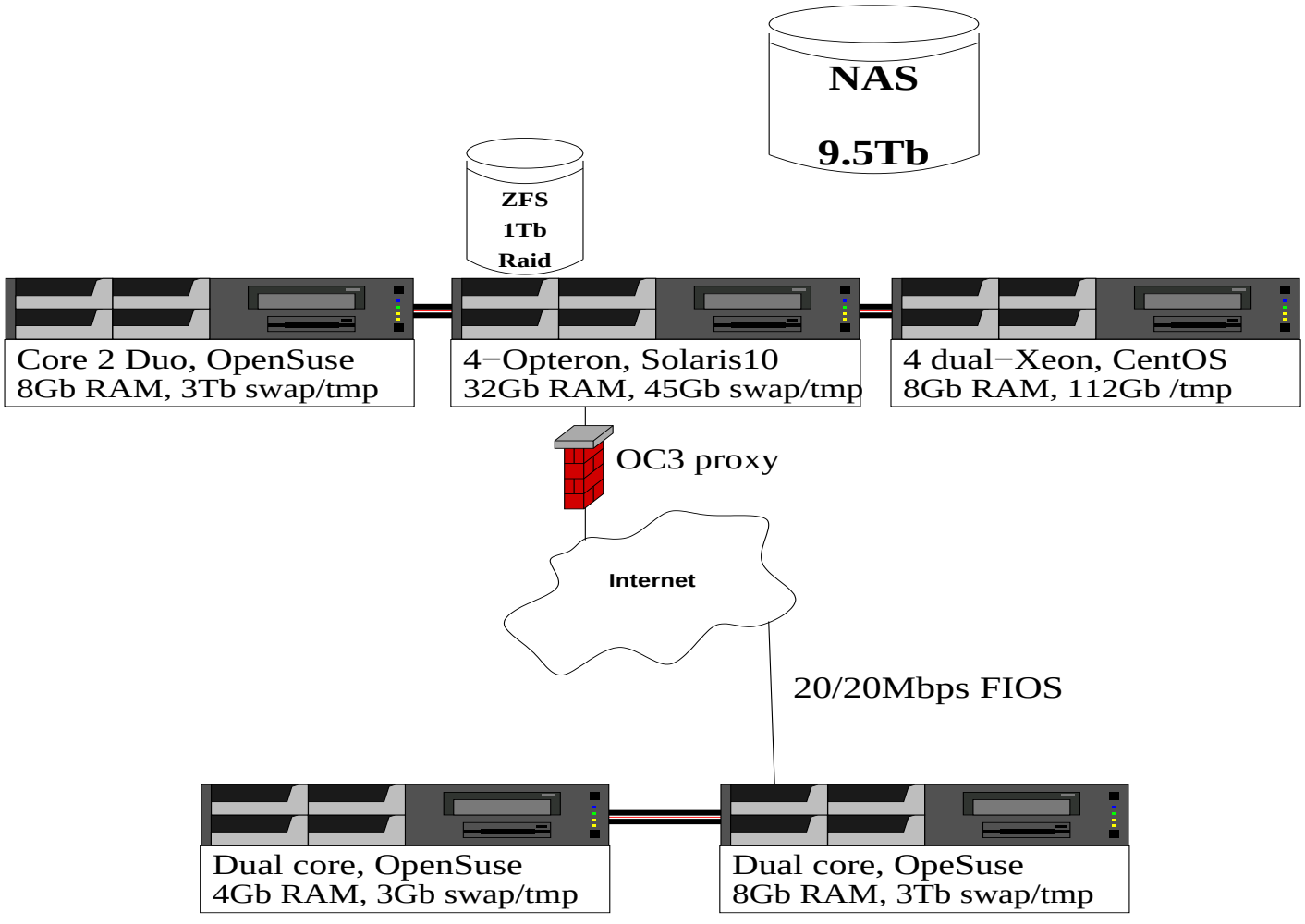
- Create a spider utilizing a search engine, and seeded by project directories (RawMeat, FSF) to grab these URLs from projects' home page
  - Search for VCS-specific URL patterns
    - cvs[:.], svn[:.], git[:.], hg[:.], bzr[:.]
    - Entice projects themselves to submit a pointer to their VCS by providing a compelling service (licensing, origin, quality)
- Example discovery/update challenge
  - gitorious.org went from 68 web pages listing projects in Jan 4, 2009 to 98 last week and changed the home page format

Copy, log, extract

	URL pattern	Clone repository	List revisions
CVS	d:pserver:user@cvs.repo.org:/	rsync	cvs log
Subversion	{svn,http}://PRJ.repo.org/	svm sync URL	svn log -v URL
Git	git://git.repo.org/	git clone URL PRJ	git log OPTIONS
Mercurial	hg://hg.repo.org/	hg clone URL	hg log -v
Bazaar	http://bzr.repo.org/	bzr branch URL	

	Extract content
CVS	rcs -pREV FILE
Subversion	svn cat -rREV URL/FILE@REV
Git	git show REV:FILE
Mercurial	hg cat -rREV FILE
Bazaar	bzr cat -rREV FILE

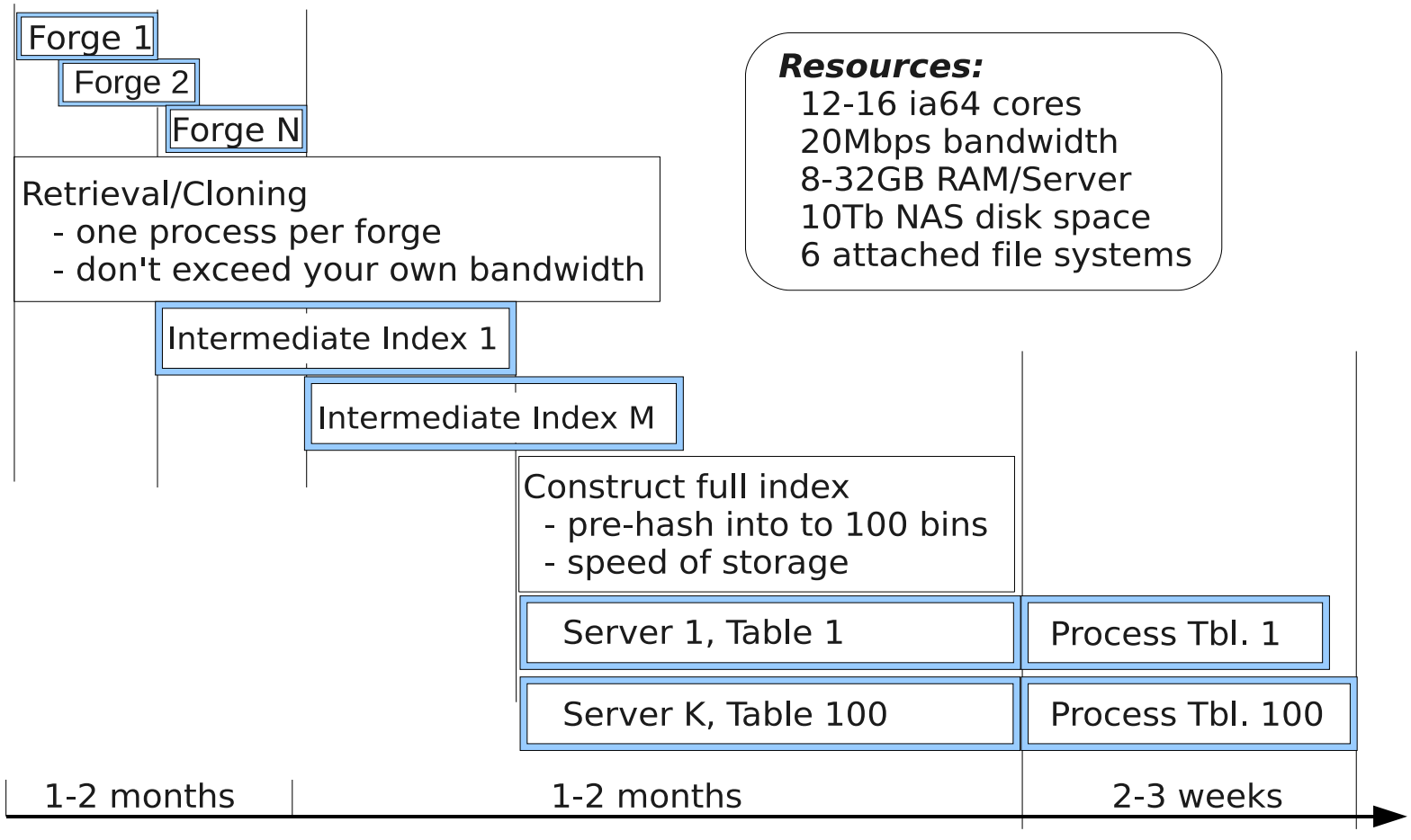
Existing system



Job scheduling

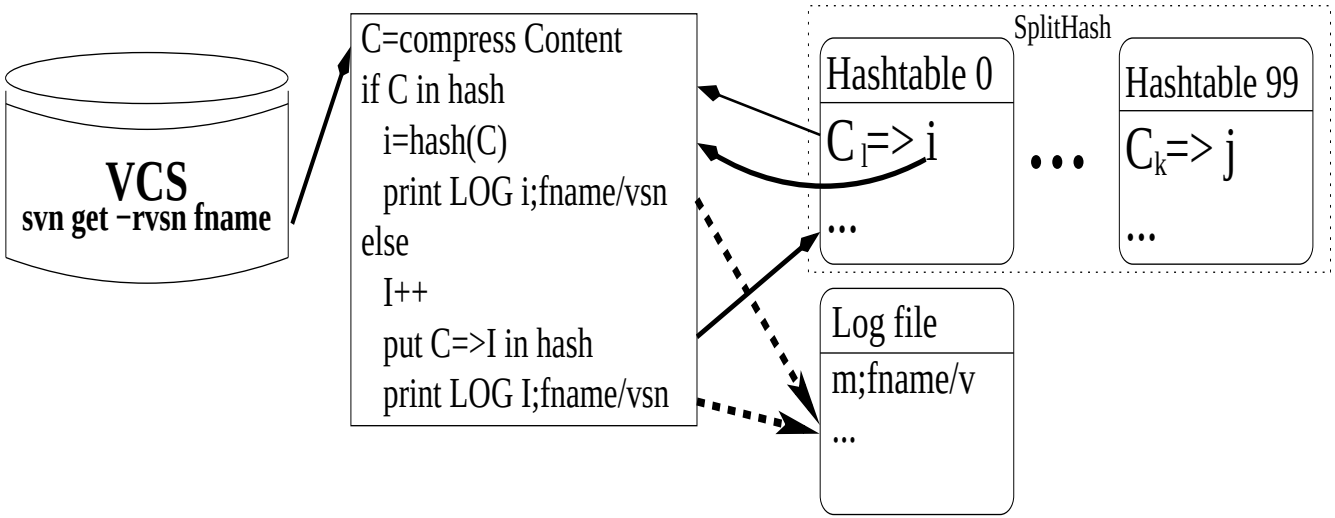
- Retrieval/cloning: No more than one(three) process per forge/repository (ethics)
- Extraction: as some cloned repositories become available use all available processors (processing time), store content in intermediate hashtables to avoid bottleneck of a single table
- Indexing: as intermediate hashtables become available distribute (via pre-hashing) to multiple servers/file/systems to store it
- Further processing: trigrams, AST, do in parallel on all available servers after the main hashtable (composed of 100 tables) is complete

Job scheduling: Gantt Chart



Extract/Index: store/index content

- Compress: $ccon = compress $con
- Insert into a hashtable (DBFile): $clones{$ccon} = $idx
  - where $idx is a new integer if content is not in the table
  - $clones{$ccon} if such content already exists
- print log: $idx;size;$FILE/$VERSION



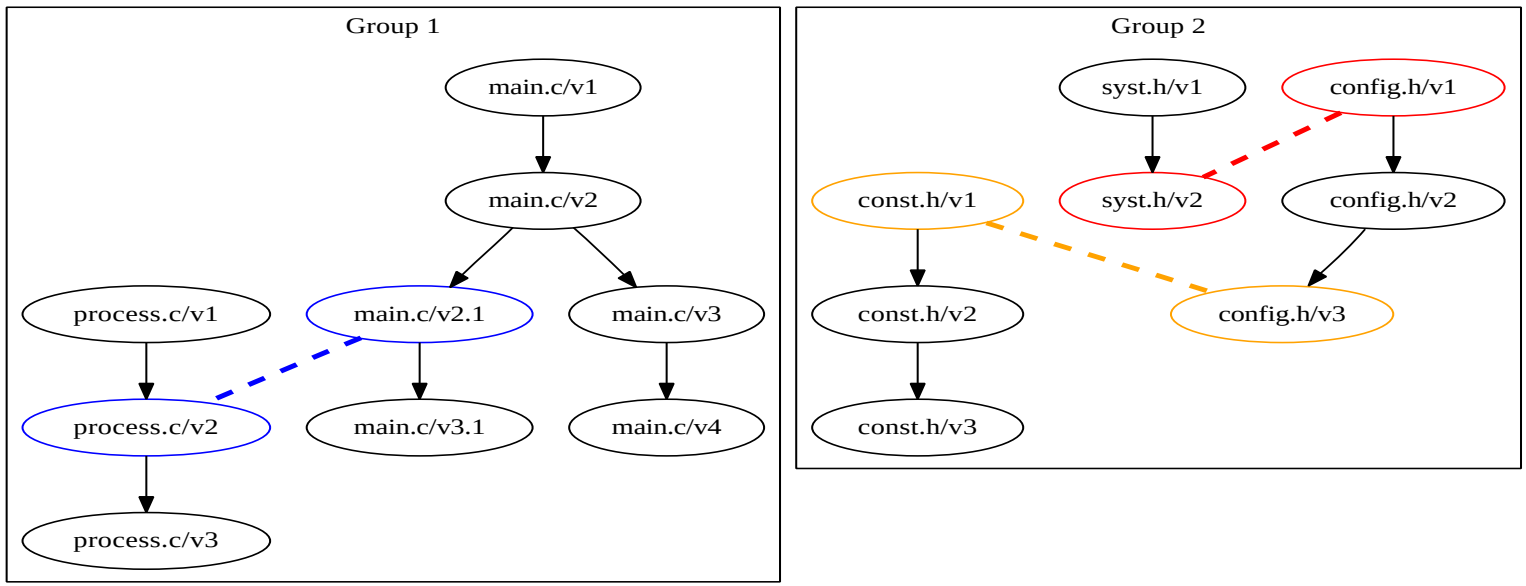
What is out there?

Forge	Type	VCSs	Files	File/Ver.	Uniq.	Space	From
Large cmpny.	Var.	>200	3,272K	12,585K	4,293K	remote	1988
SourceForge	CVS	121K	26,095K	81,239K	39,550K	820GB	1998
code.google	SVN	42K	5,675K	14,368K	8,584K	remote	1996
github.com	Git	29K	5,694K	18,986K	7,076K	154GB	1988
repo.or.cz	Git	1,867	2,519K	11,068K	5,115K	43GB	1986
gitorious.org	Git	1,098	1,229K	4,896K	1,749K	20GB	1988
Debian	Git	1,662	1,058K	4,741K	1,863K	19GB	1988
Savannah	CVS	2,946	852K	3,623K	2,345K	25GB	1985
objectweb.org	CVS	93	1,778K	2,287K	528K	17GB	1999
SourceWare	CVS	65	213K	1,459K	761K	10GB	1989
netbeans	Hg	57	185K	23,847K	492K	69GB	1999
rubyforge.org	SVN	3,825	456K	807K	256K	4.9GB	2001
Mozilla	Hg	14	58K	210K	105K	1.6GB	2000

Forge	Type	VCSs	Files	File/Ver.	Unique F/V	Space	From
git.kernel.org	Git	595	12,974K	97,585K	856K	205GB	1988
OpenSolaris	Hg	98	77K	1,108K	91K	9.7GB	2003
FreeBSD	CVS	1	196K	360K	75K	2.5GB	1993
Kde	SVN	1	2,645K	10,162K	527K	50GB	1997
gnome.org	SVN	566	1,284K	3,981K	1,412K	1GB	1997
Freedesktop	CVS	75	139K	784K	375K	4GB	1994
Gcc	SVN	1	3,758K	4,803K	395K	14GB	1989
Eclipse	CVS	9	729K	2,127K	575K	11GB	2001
OpenJDK	Hg	392	32K	747K	60K	15GB	2008
Mysql-Server	Bazaar	1	10K	523K	133K	6GB	2000
PostgreSQL	CVS	1	6K	108K	105K	0.5GB	1994
ruby-lang	SVN	1	163K	271K	56K	0.6GB	1998
Perl	Git	1	11,539	103K	42K	0.2GB	1988
Python	SVN	1	8K	89K	76,454	0.8GB	1991

How to construct Universal Version History?

- Establish links among files across multiple VCS
  - identical content: the closure of files sharing at least one identical version
  - Also: identical AST, Trigram, other ways to establish identity or similarity



Implications

- Census is possible with just 4 servers
- Discovery/Update challenges
  - Brute force — a better spider
  - Carrot — compelling applications for projects to register
- VCS challenges — move to Git! (though still has no decent GUI)
  - Add a function to extract all content (version-by-version too slow)
  - Add and use author (in addition to commiter) field
  - Identify all parents of a change
- What services to provide?
  - Too big to copy — process in place
  - Start with: code origin, quality, reuse [1]

References

Audris Mockus.
Large-scale code reuse in open source software.
In *ICSE'07 Intl. Workshop on Emerging Trends in FLOSS Research and Development*, Minneapolis, Minnesota, May 21 2007.

Richard W. Selby.
Enabling reuse-based software development of large-scale systems.
IEEE Trans. on Software Engineering, 31(6):495–510, June 2005.