Using software trails to recover the evolution of software

3rd ELISA 2003

Daniel M. German
Software Engineering Group
University of Victoria, Canada

September 23, 2003 Version: 1.0.0

Introduction

- By using tools that become vital to the success of a project, its history is being recorded in *software trails*:
  - Configuration management systems (including version control and defect management systems)
  - Mailing lists
  - ChangeLogs

Evolution

- The initial objective of this research was to try to recover the evolution of Evolution using its software trails
  - It is the *Outlook* of the GNOME project
  - Almost 4 years of development
  - It is becoming one of **the** free mail clients
  - Unlike many other OSS projects
    - It started as a *group* project, with its software requirements drawn before the code was written
    - It has been driven by one company: Ximian (recently bought by Novell)

Methodology

- Define a schema that represents and correlates software trails
- Gather the trails:
  - Recover the trails and map them to the schema
  - Trails are usually available as logs and history reports
- Extend the information:
  - Combine the available information, creating new facts
  - It might require some heuristics
- Analyze:
  - Using query languages and visualization tools
  - It is a time consuming task

Is this info useful?

- The most important question: **can we trust this information**?
- The answer: **it depends**
- Some projects establish clear guidelines –and follow them– on how to use these tools.
  - IBM uses a Configuration Management System that tracks several trails
  - Many free/Open Source software projects use a toolkit based on CVS, Bugzilla, mailman, following a set of de-facto standards

Evolution **Trails**

- This papers uses info from
  - ChangeLogs: “explain how earlier versions of software were different from the current version.”
  - CVS: Most popular version control system
    - ∗ Keeps track of who modifies what, and when, supports branching
    - ∗ It does not support transaction-oriented operations
  - Mailing lists
    - ∗ For developers and for users
  - Source code releases
- In several cases, it was necessary to reverse engineer their formats

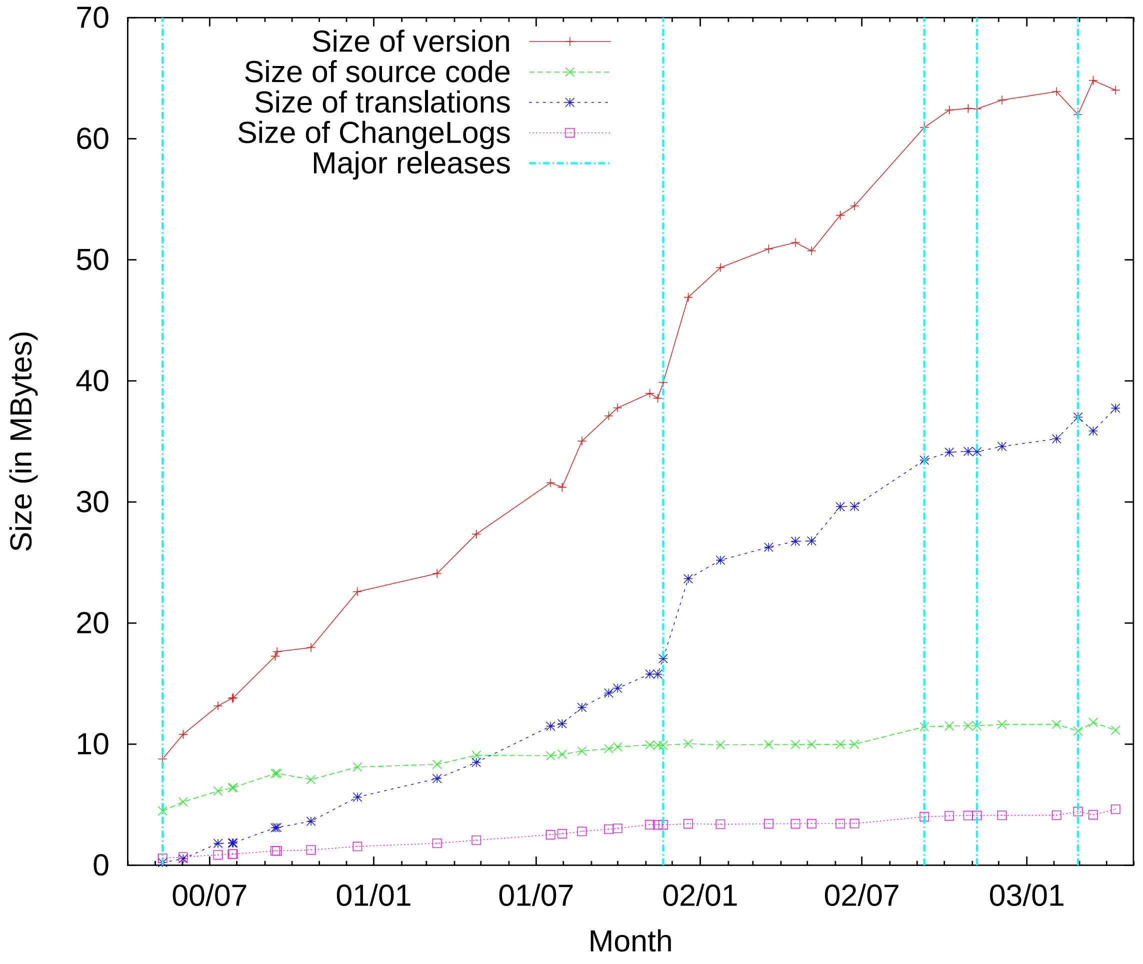
The Challenge of Extending the Trails

- It is difficult to correlate *raw* trails
- For example, identifying developers:
  - CVS uses an *id* to record the developer
  - The ChangeLog lists his/her preferred email address
  - The mailing list might list his/her spam, or commonly used address
  - Some changes come from non-cvs developers and they are recorded in the ChangeLogs
- Nonetheless, they provide a gold mine of information to follow the evolution of a project

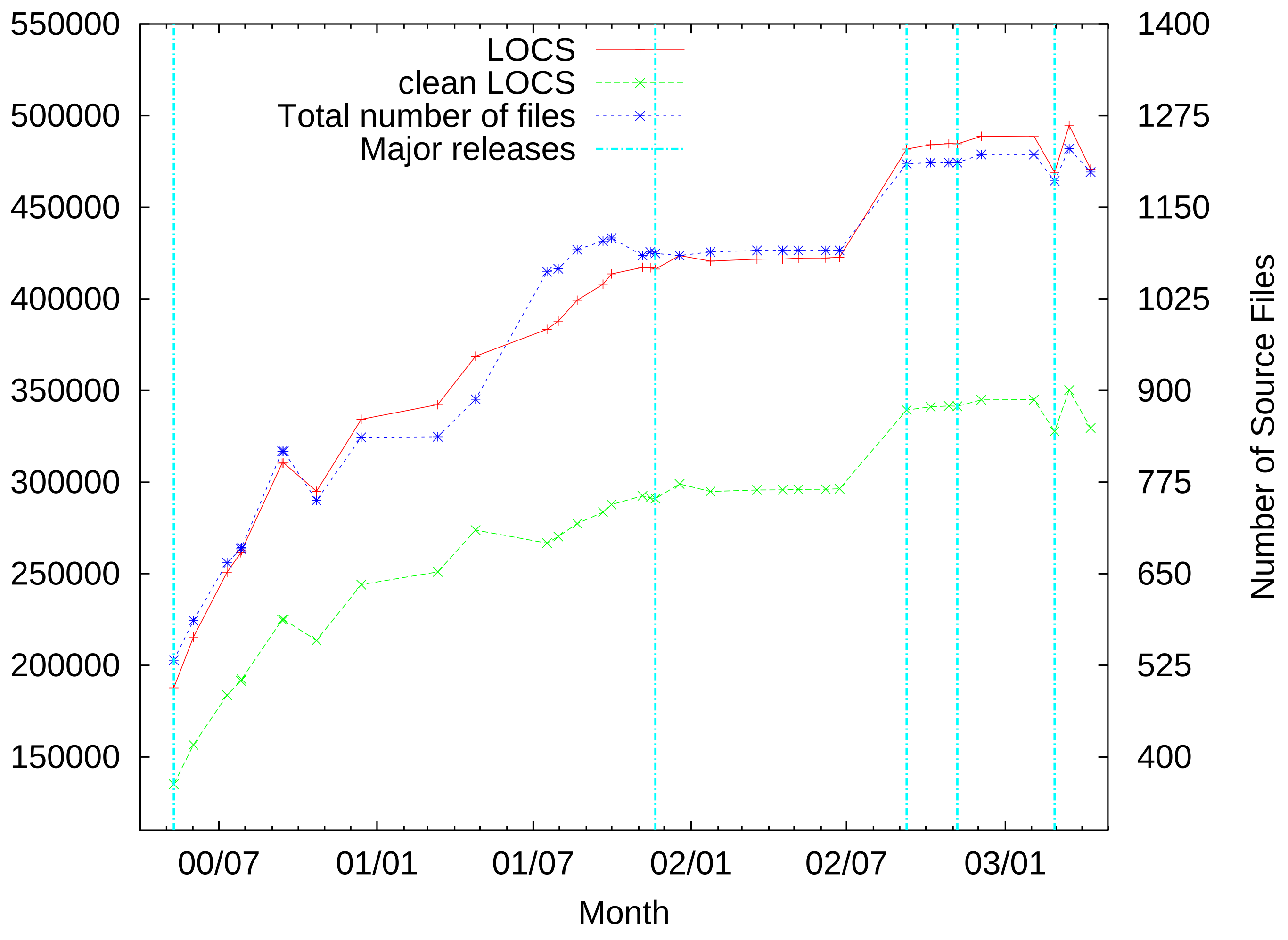
Milestones of Evolution

Milestones	Date
Coding of camel starts	1999-01-01
Evolution starts	1999-04-16
Ximian is established	1999-10-01
Version 0.0	2000-05-10
Version 1.0	2001-11-21
Version 1.1.1	2002-09-09
Version 1.2.0	2002-11-07
LinuxWorld “Best Front Office Solution” award	2003-01-23
Version 1.3.1	2003-02-28

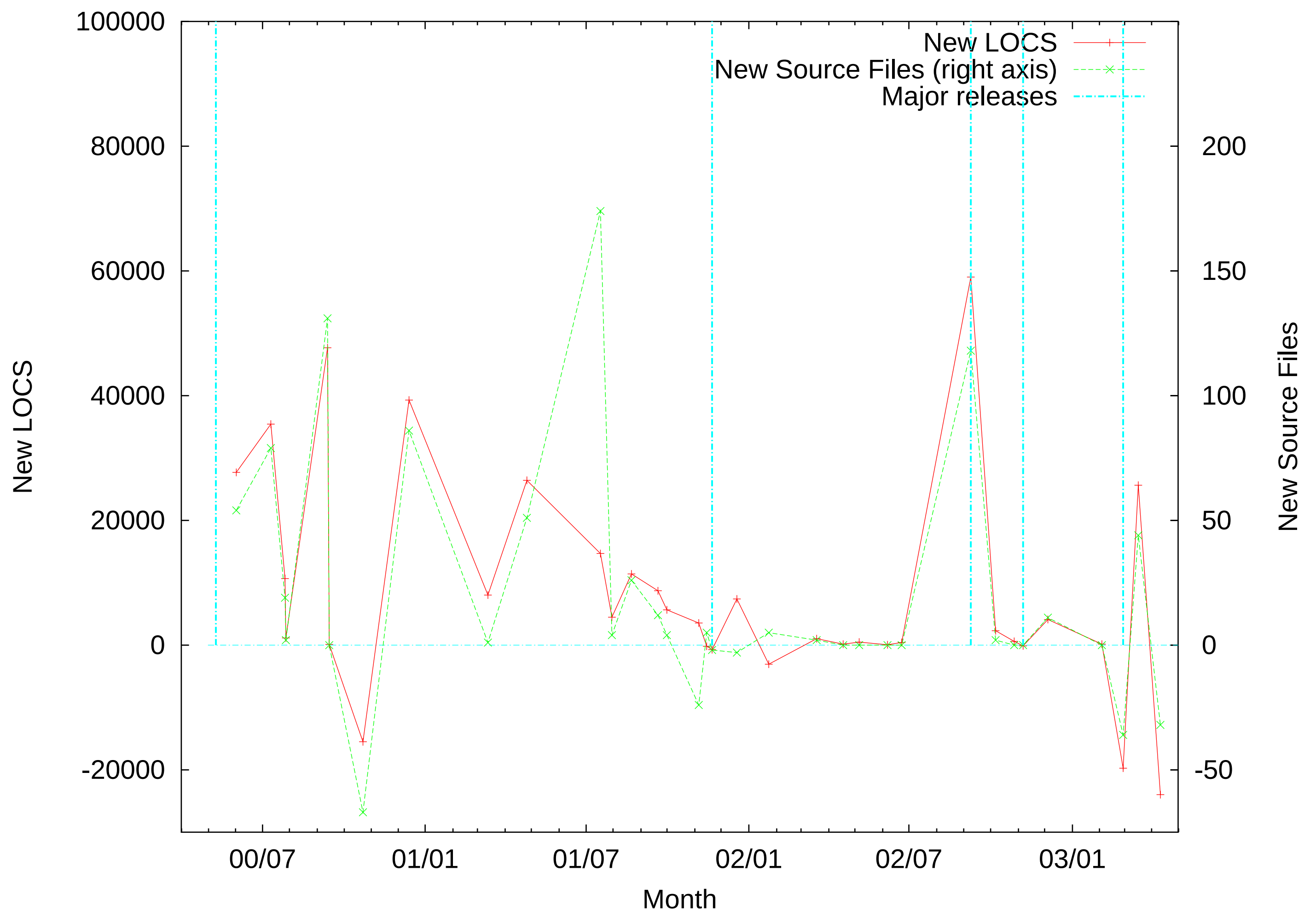
Size of the Distributions



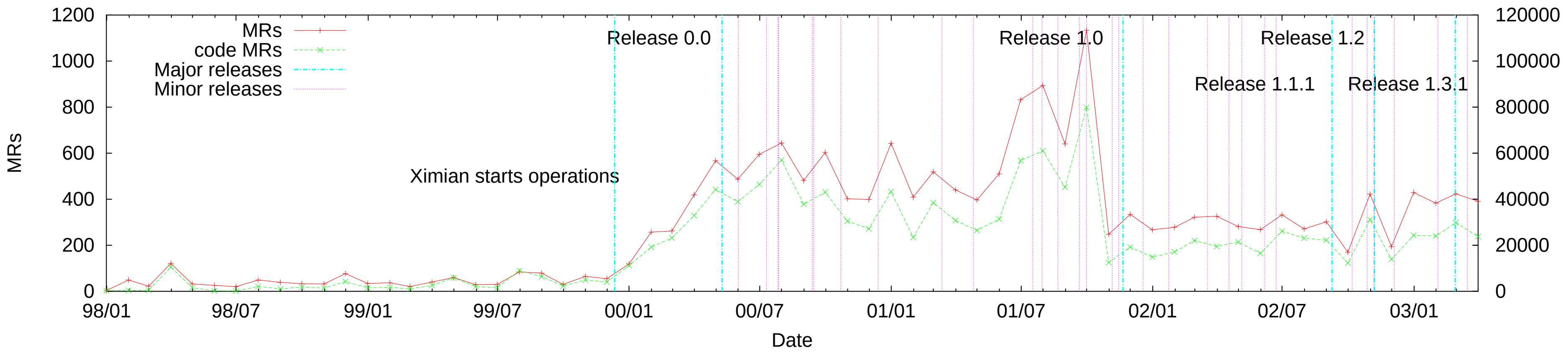
Size of the Distributions...



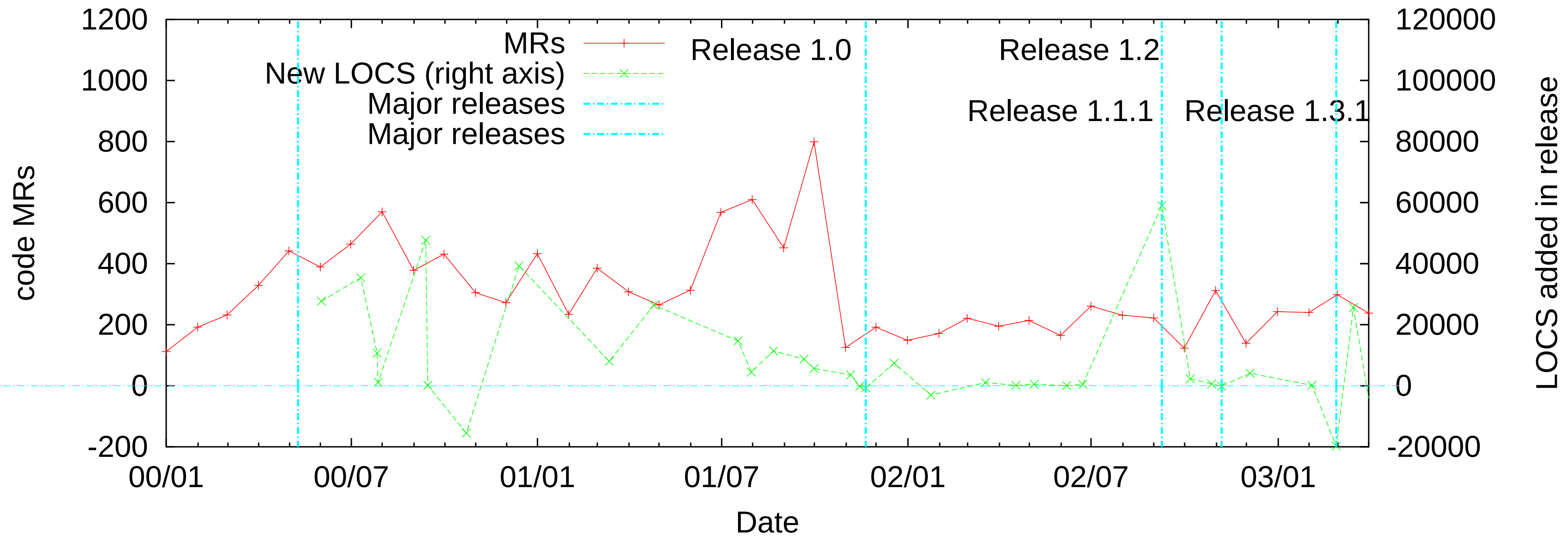
How is the code base changing?



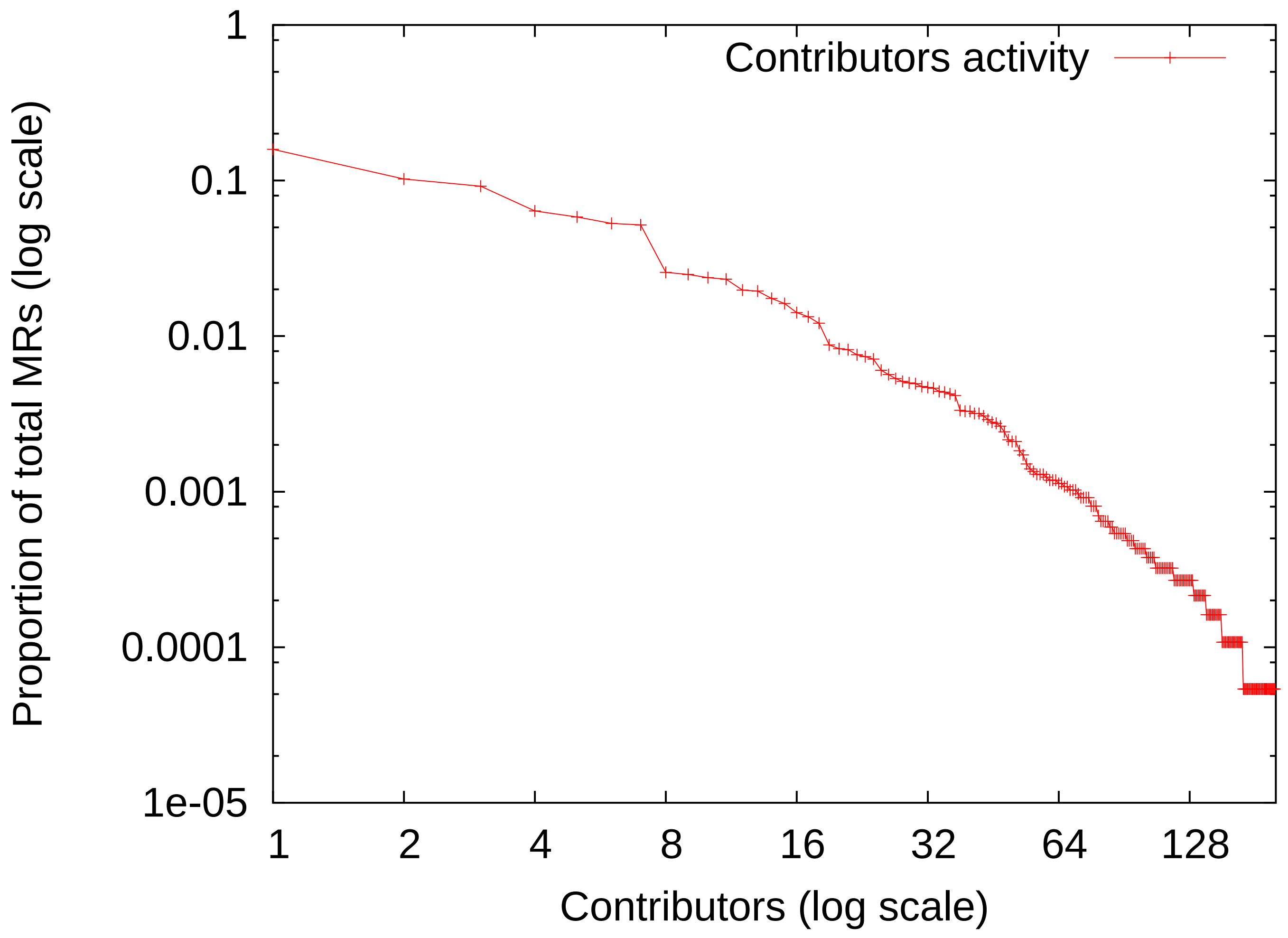
And the developers?



Change in code base vs. contributors activity



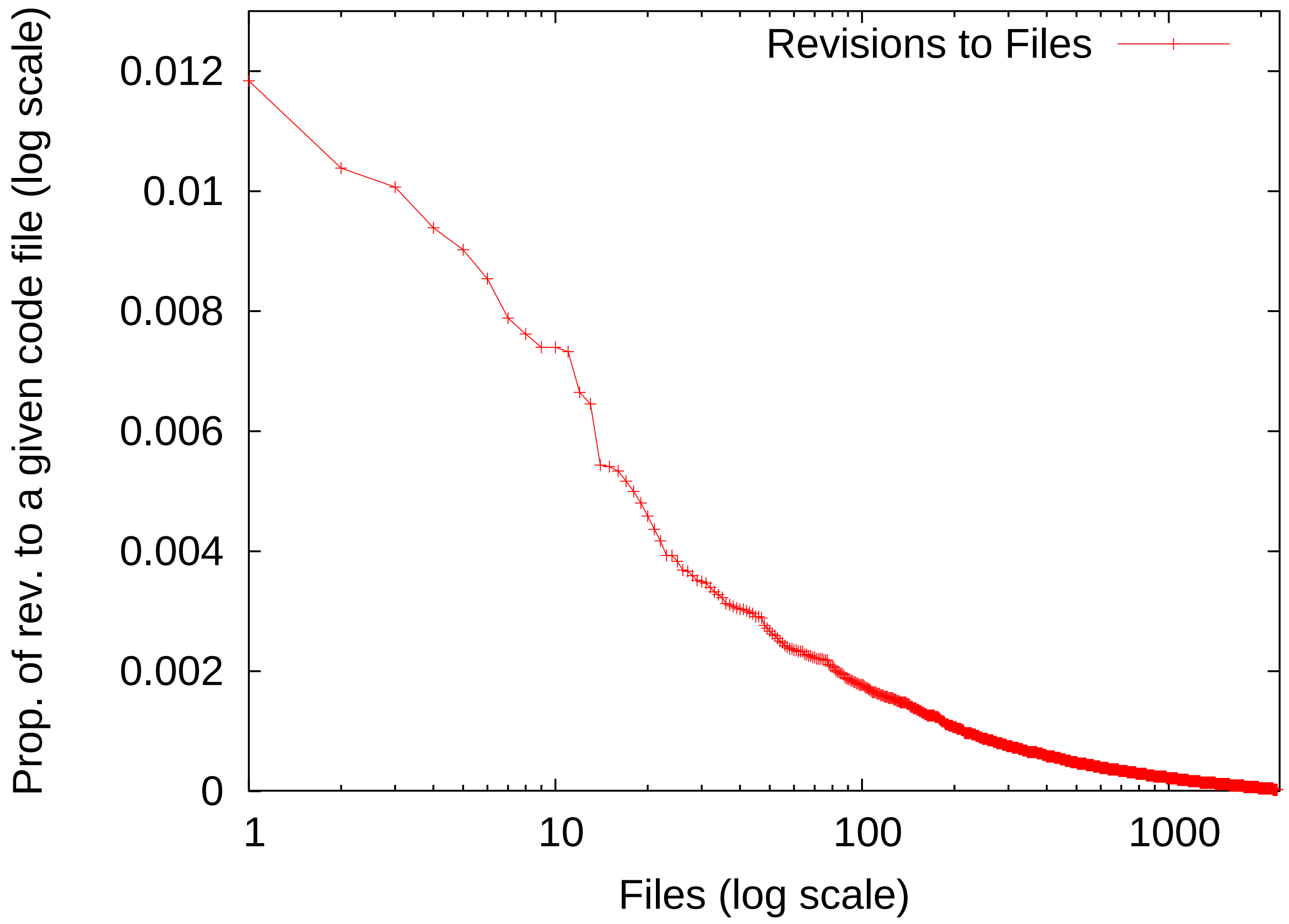
How many contributors?



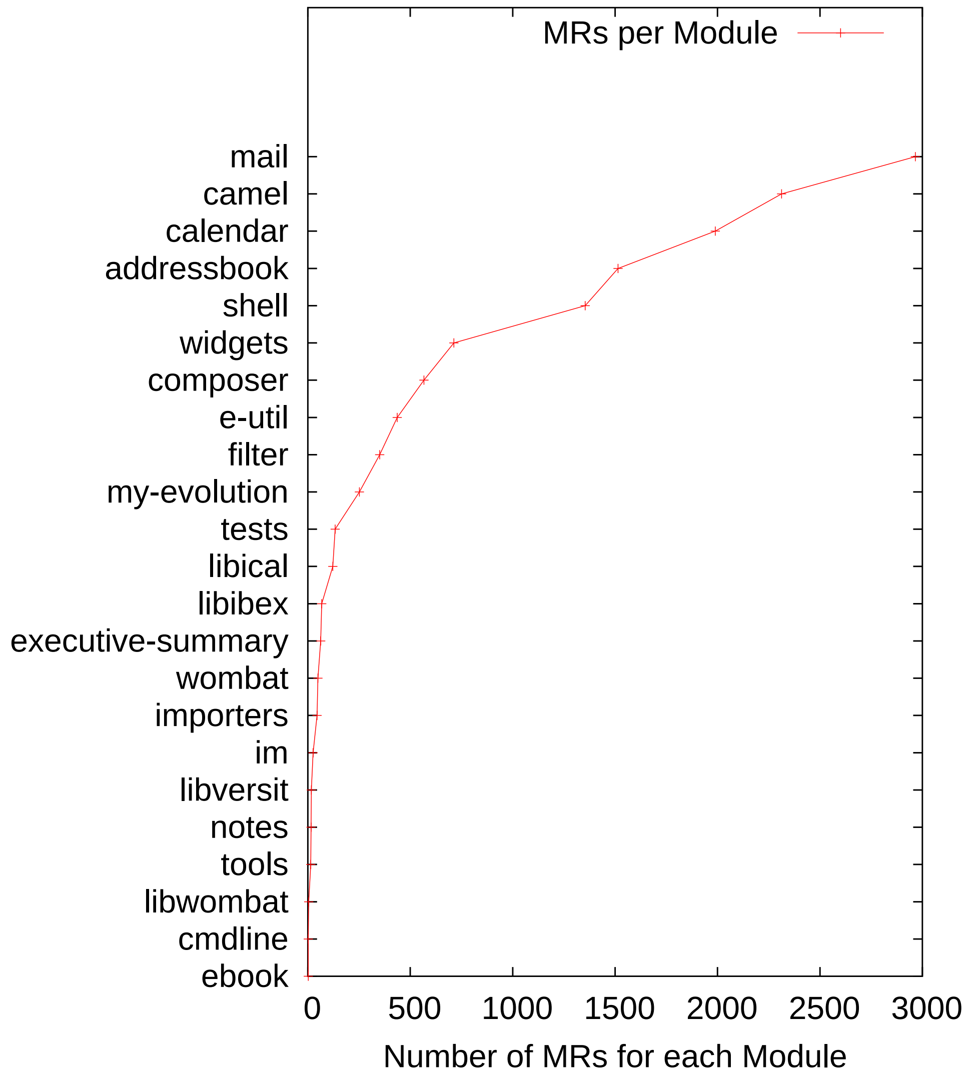
Revisions per type of file

Extension	Prop.	Accum.	Number of files in CVS
.c	0.41	0.41	1195
ChangeLog	0.22	0.62	43
.h	0.13	0.75	1063
.am	0.05	0.81	174
.po	0.04	0.85	71

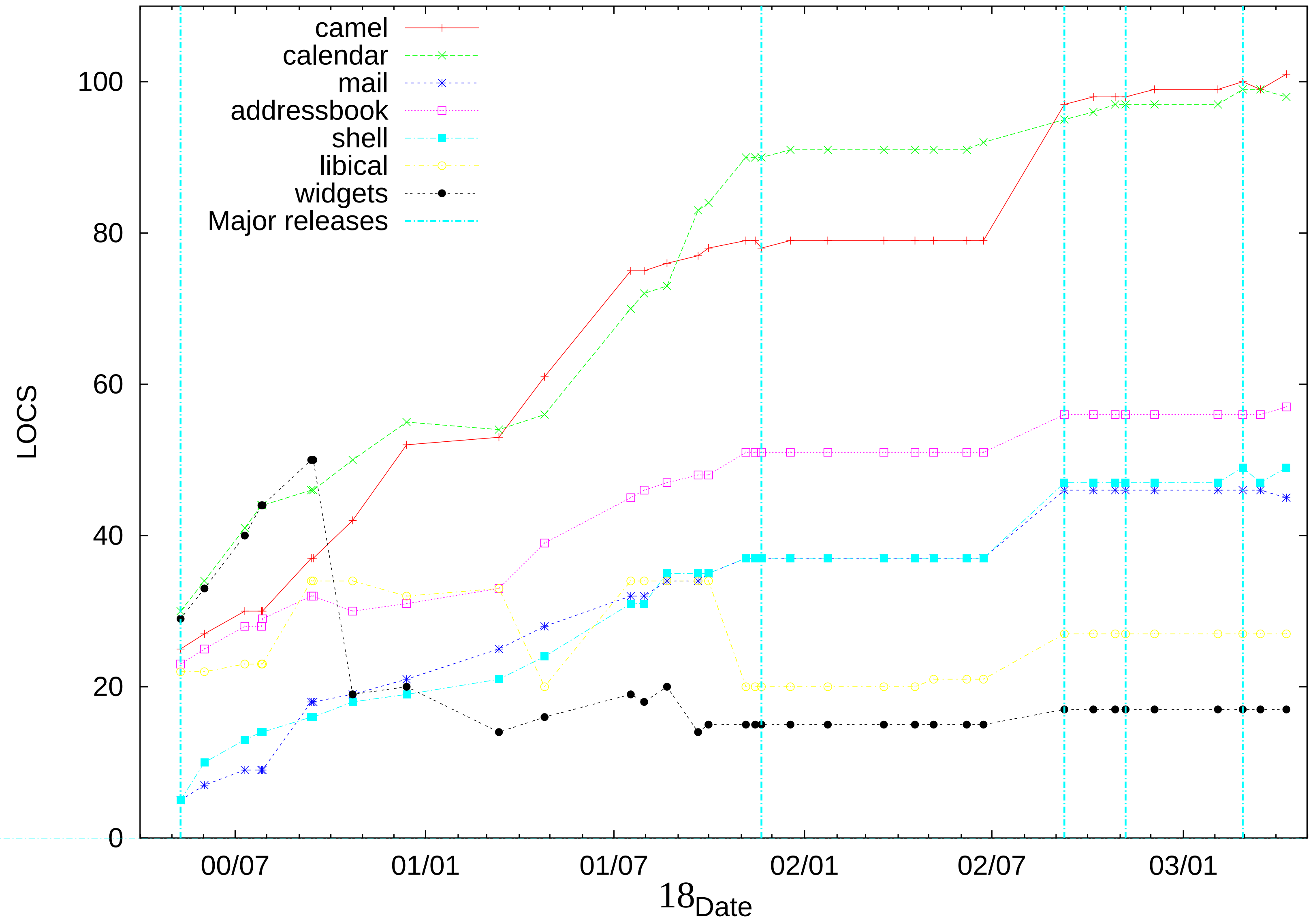
Most files are rarely changed



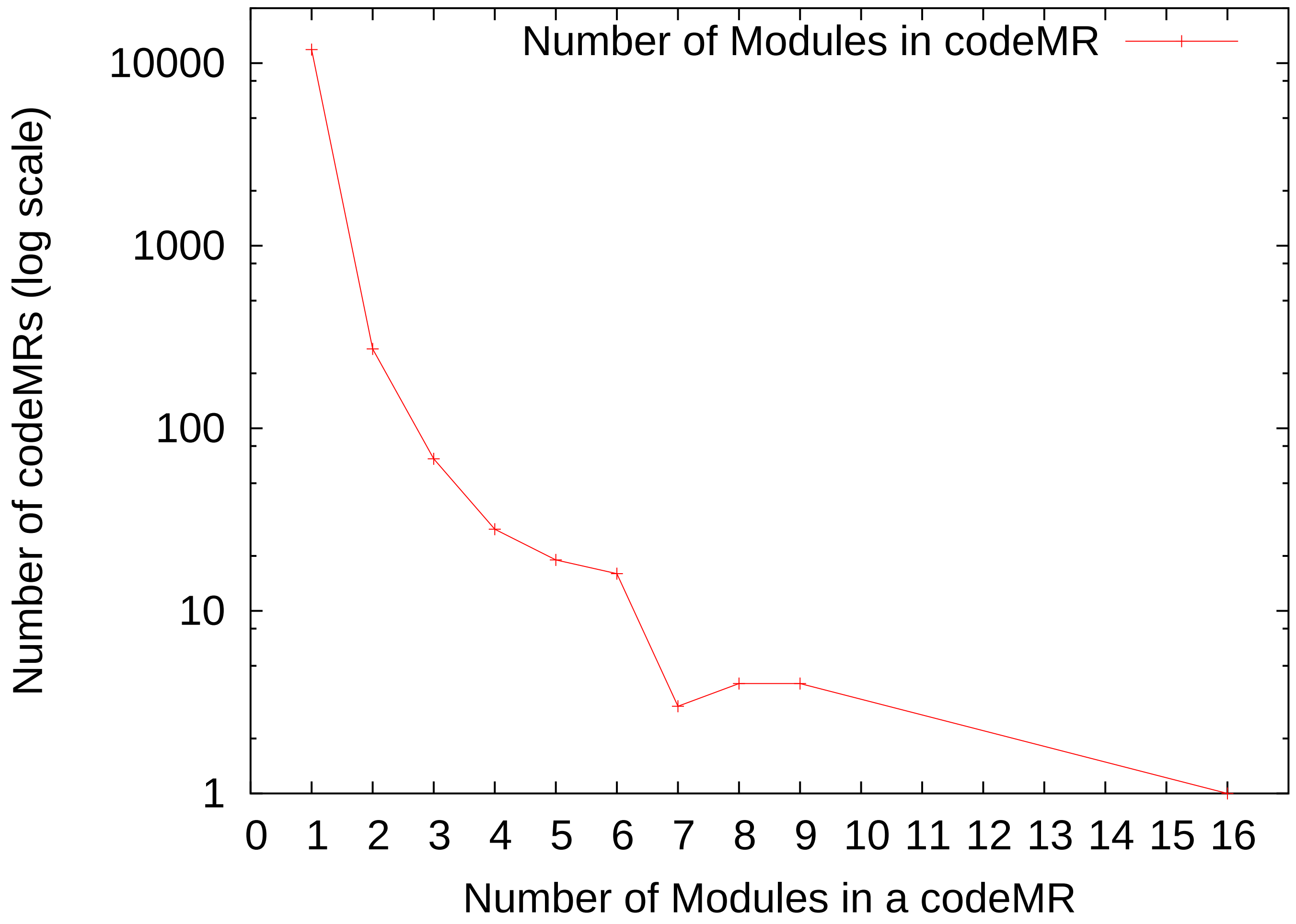
Modules


MRs per Module
mail
camel
calendar
addressbook
shell
widgets
composer
e-util
filter
my-evolution
tests
libical
libibex
executive-summary
wombat
importers
im
libversit
notes
tools
libwombat
cmdline
ebook
0
500
1000
1500
2000
2500
3000
Number of MRs for each Module

Evolution of the size of the modules



Changes are usually localized in a given module



Developers tend to concentrate in one module

Mod	Developers	Id	Prop	Acc
shell	17	ettore	0.65	0.65
		danw	0.11	0.76
		toshok	0.05	0.81
		clahey	0.04	0.84
		zucchi	0.03	0.87
mail	19	fejj	0.52	0.52
		rodo	0.13	0.65
		zucchi	0.12	0.77
		ettore	0.07	0.83
		danw	0.06	0.89
calendar	17	jpr	0.40	0.40
		rodrigo	0.32	0.72
		ettore	0.07	0.79
		danw	0.06	0.85
		damon	0.03	0.88

Observations

- One software trail does not tell the whole story
- Schema evolution
- Informal structure in trail
- Information overload and the need for analysis and visualization tools.
- Quality of software trails.

Quality of Trails

- Some projects keep better trails than others.
- One hypothesis: it is a measure of:
  - The number of developers,
  - their dislocation,
  - and the maturity of the project.

Conclusions and Future Work

- Extracting and correlating software trails can tell a detailed story of how a software project has evolved
- But it comes at a cost: too much information to analyze
- It is needed:
  - Creating of standardized schemas
  - More tools to recover and enhance the trails
  - Heuristics to automatically discover “interesting” facts
  - Metrics to quantify trails