
Getting the global picture

*Jesús M. González Barahona, Gregorio Robles
GSyC, Universidad Rey Juan Carlos, Madrid, Spain*

{jgb,grex}@gsyc.escet.urjc.es



*Oxford Workshop on Libre Software 2004
Oxford, UK, June 25th*

Overview

- Available information about libre software projects
- Open problems (large detailed studies, crossing information from different sources)
- Some questions still to be answered
- What do we need to be there

Sources of data about libre software projects

- Version control systems: CVS, Subversion, Bitkeeper, etc
- Software releases (both binary and source)
- Project documentation: man, info, DocBook, LaTeX, plain text, etc
- Bug tracking systems: Bugzilla, Sourceforge, Debian, etc
- Mailing lists: BSD mbox, MH mbox, Mailman, etc
- Forums: many, many kinds
- Information about usage, eg: Debian's popularity contest
- Impact in the Internet, eg: some filtered Googling
- Polls and surveys, eg: FLOSS

One kind of data source, one project

Example: source code analysis

- Data source: version control system
- Metrics based analysis (SLOC, McCabe, number of modules, etc.)
- Classification of code (language, documentation, etc.)
- Reuse study (comparison of source code)
- Contribution (eg. by author), including affiliation networks
- Evolution (any of the previous in time)
- Combined studies (within same project)
- What can be learned: structure of the source code, basic developer activity

Several kind of data sources, one project

Example: tracking developer activities

- Data source: version control system, bug tracking system, mailing list
- Identify all developers in the BTS (maybe with help of heuristics)
- Identify all BTS ids in mailing lists (maybe with help of heuristics)
- Track individual developers in time (evolution of their contribution to the project)
- What can be learned: how activity evolves over time, who fix bugs (and when), ratio of listers to reporters to developers

One kind of data source, several projects

Example: source code analysis for a distribution

- Data source: source packages in a distribution
- Compare and correlate source analysis (already shown)
- What can be learned: file size for different languages, correlations between metrics and developers (are they similar in similar projects?)

Several kinds of data sources, several projects

Example: relationship of bug fixing to patch size

- Data sources: version control system, bug tracking system, mailing list
- Look for patches in the BTS, identify them in the CVS
- Look for patches in the mailing list, identify them in the CVS
- Look for fixed bugs in the BTS, relate them to changes in CVS
- What can be learned: time from bug report to bug fix, relationship to patch size, to who takes the bug report, to existence of patch: relationship of bugs to previous changes in code

Several kinds of data sources, thousands of projects

Example: tracking developer effort and activities

- Data sources: as much as possible
- Select some hundreds of developers
- Track them in several projects, submit a poll to them
- Use the combined information to estimate effort per developer over time, to look for shifts in effort from project to project, to correlate effort in different activities (coding, bug fixing, mailing lists)
- What can be learned: typical evolutions of developers, what they think they do compared to what they actually do, understanding why some projects get developers and other no, and model the project

In all cases...

- All the downloading of data can be automated
- Most of the analysis of data can be automated (maybe with the help of heuristics statistically valid, and some hand-work)
- Data can benefit a lot of well designed polls answered by developers
- Really large sets of data
- Many privacy issues

Main problems to get the big picture

- Different source systems (eg, bug tracking systems: Bugzilla, Sourceforge, GNATS, Debian)
- Different levels of information and data representation for the same concept (eg: user ids in CVS, BTS, mailing list, forum, etc)
- Different information for the same item (eg: different mailing addresses for the same developer, at the same time)
- Different conventions at different projects (eg: policy and uses of code uploads and releases)

Our plan (headlines)

- Automate as much download processes as possible (modular architecture)
- Automate as much analysis approaches as possible (modular architecture)
- Build huge database with all data collected (be it raw or result of analysis)
- Allow other to use an contribute code
- Allow data from polls to be integrated
- Run the machinery for many, many projects

Our plan (some details)

- Unique (opaque) identifier for every developer
- Unified data descriptions for main sources of raw data
- Clear data formats for exchange of information in most common contexts
- Let projects use the tools (“report on my project”)
- Integrate the tools with usual development systems (as GForge)

Where are we now

- GlueTheos and CVSanaly in good shape
- Work in progress: integration with source analysis tools
- Work in progress: integration of social network analysis tools
- Work in progress: integration with statistical tools
- To be done: integration of other data sources (BTSs, mailing lists, etc)
- To be done: framework for data interchange
- To be done: integration of everything, and collaboration framework

CVSanaly: analyzing CVS repositories

- Based on the analysis of CVS logs
- Three steps
 - Preprocessing (data retrieval and extraction)
 - Intermediate format (SQL, XML...)
 - Postprocessing (manipulation, correlations, graphics, etc.)

Preprocessing

- Downloading modules and removing aggregated ones
- Log retrieval and parsing
- Transformation into SQL and XML
- Username merging
- File type matching (source code, documentation, translation, etc.)
- 1.1.1.1 version and files in the Attic
- Commit comment parsed for external contribution and “silent” commits

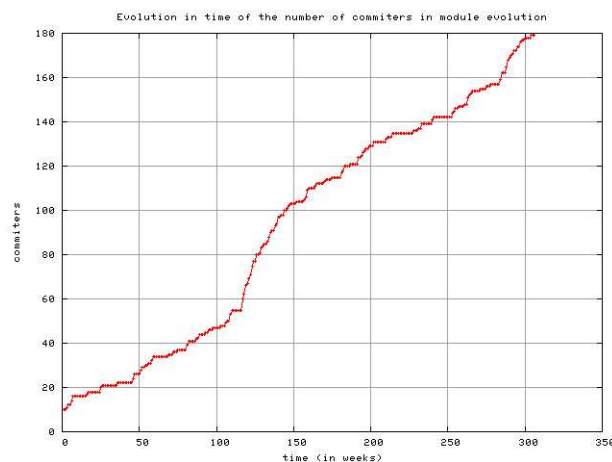
Postprocess

- Statistical information on the project
- Software evolution, inequality, etc. graphs
- Heat maps for developer interaction
- Social Network Analysis (for modules/directories/files and developers)
- Developer statistics

GlueTheos: analysis of the evolution

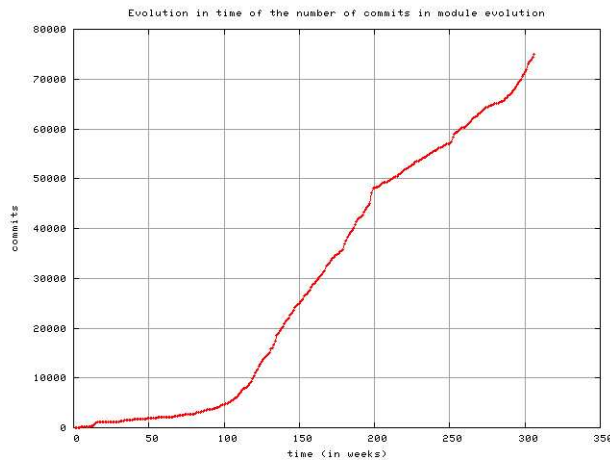
- Retrieves periodically the sources from CVS
- Runs external programs
 - Size measurement in SLOC (SLOCCount)
 - Authorship attribution (CODD)
 - Complexity measures (Halstead & McCabe)
 - Other (even language-specific) tools are also possible (wc, etc.)
- Stores results in database in order to make comparisons possible

Committers in time for 'Evolution'



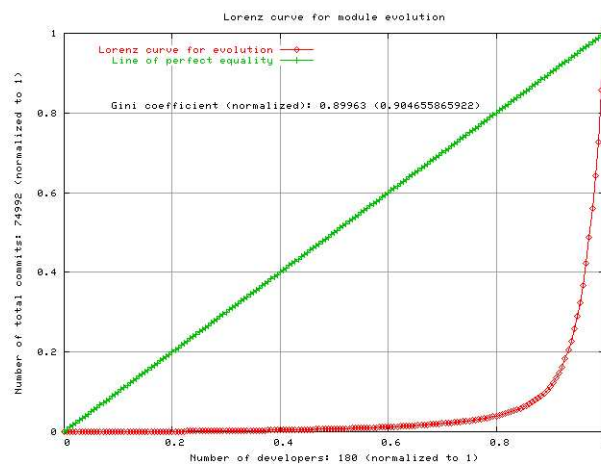
Committers in time for 'Evolution'

Commits in time for 'Evolution'



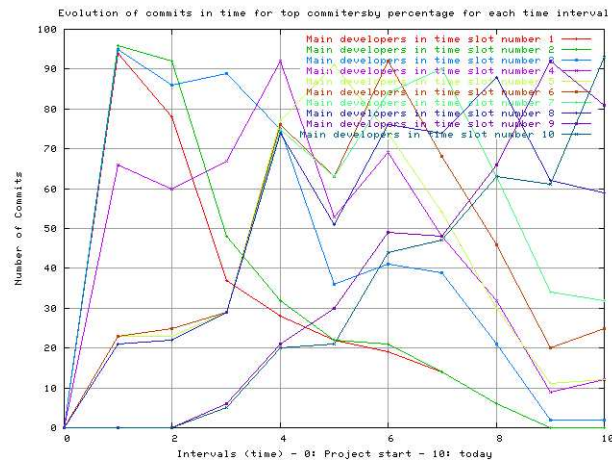
Commits in time for 'Evolution'

Gini coefficient in 'Evolution'



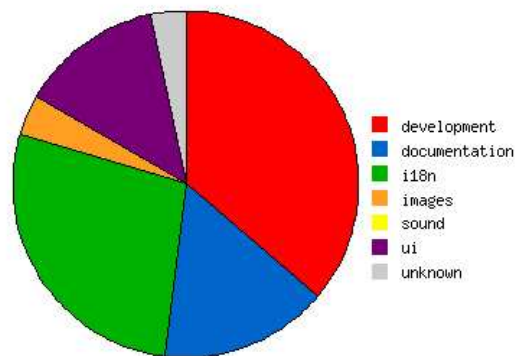
Gini coefficient in 'Evolution'

'Generations' in 'KOffice'



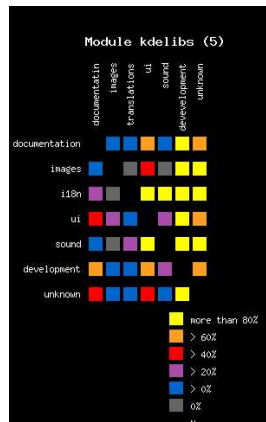
"Generations" in KOffice

File discrimination in KDE



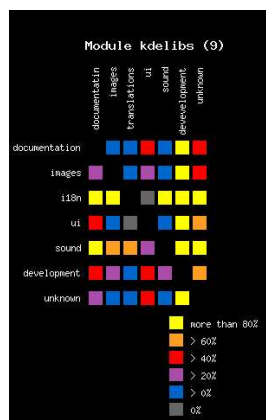
File discrimination in KDE

File discrimination by developers correlated



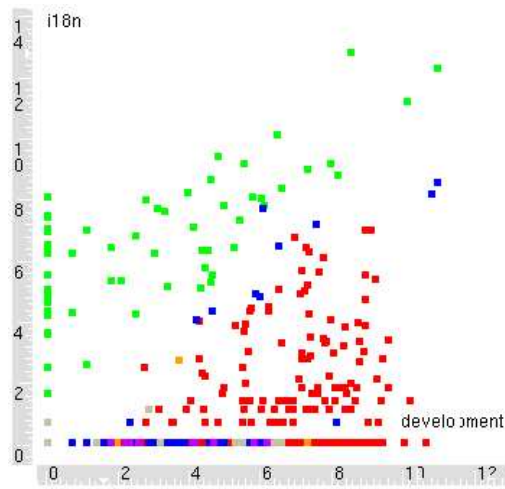
kdelibs Heatmap (5th slot)

File discrimination by developers correlated (II)



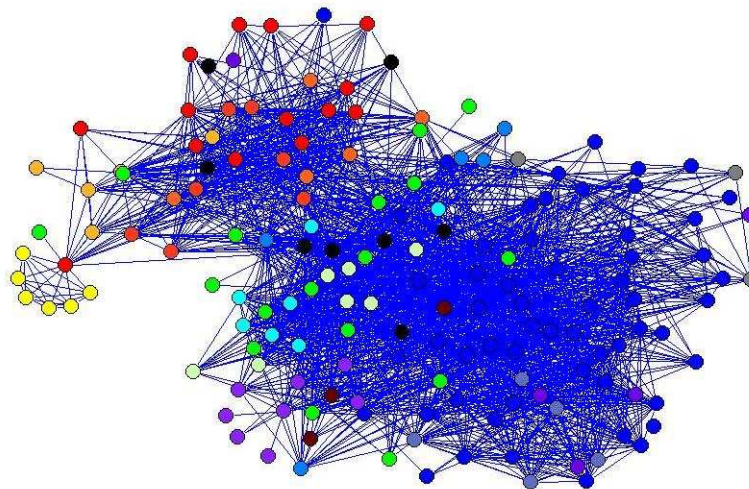
kdelibs Heatmap (9th slot)

Different communities?

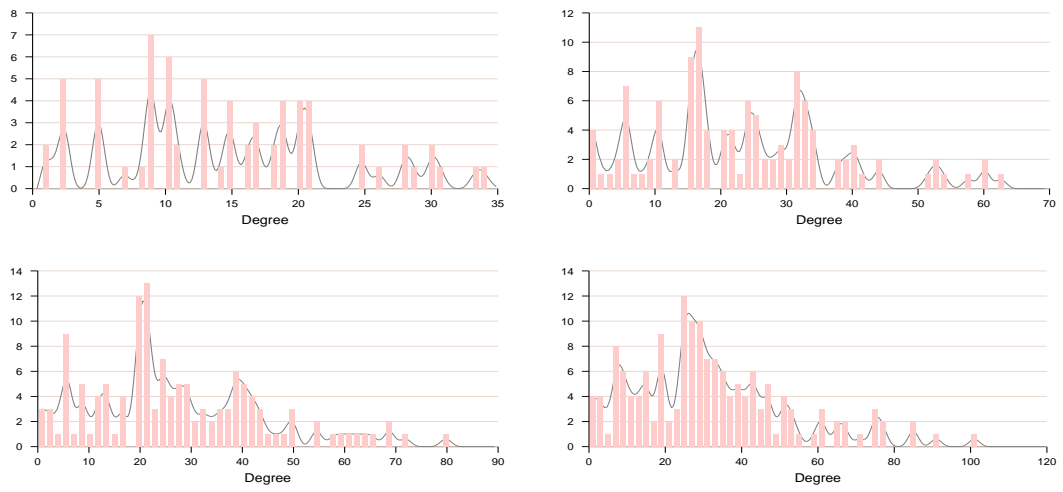


Translation vs development

The Apache modules

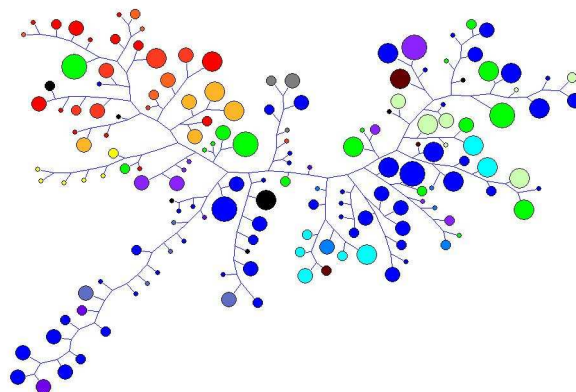


Apache connection degree (modules network)



2001 (top left) to 2004 (bottom right)

Structure of the Apache modules



Conclusions

- The field of quantitative analysis of libre software projects is maturing
- Large quantities of work can be automated
- In the end, it is a problem of data mining
- To advance with more complex studies, we need more quantities of data from different sources, and tools to handle them
- Integration with polls and surveys is fundamental

`libresoft.dat.escet.urjc.es`