

# **Free software for education, research, and sustainable development**

---

Roberto Di Cosmo

Université de Paris 7

<http://www.dicosmo.org>

Cordoba, August 10th 2007

## Free software for developing countries

---

Free software is *strategic* for developing countries :

**sustainable development** business model that favors *local* services and SMEs, instead of wealth concentration in multinational groups

**creation of qualified jobs** *locally*

Creating a software industry only requires brainpower... and a high speed connection.

**protection** against “intellectual property” zealots

**control** of the technology that is the state’s nervous system. Without free software, who knows who<sup>a</sup> is in charge ?

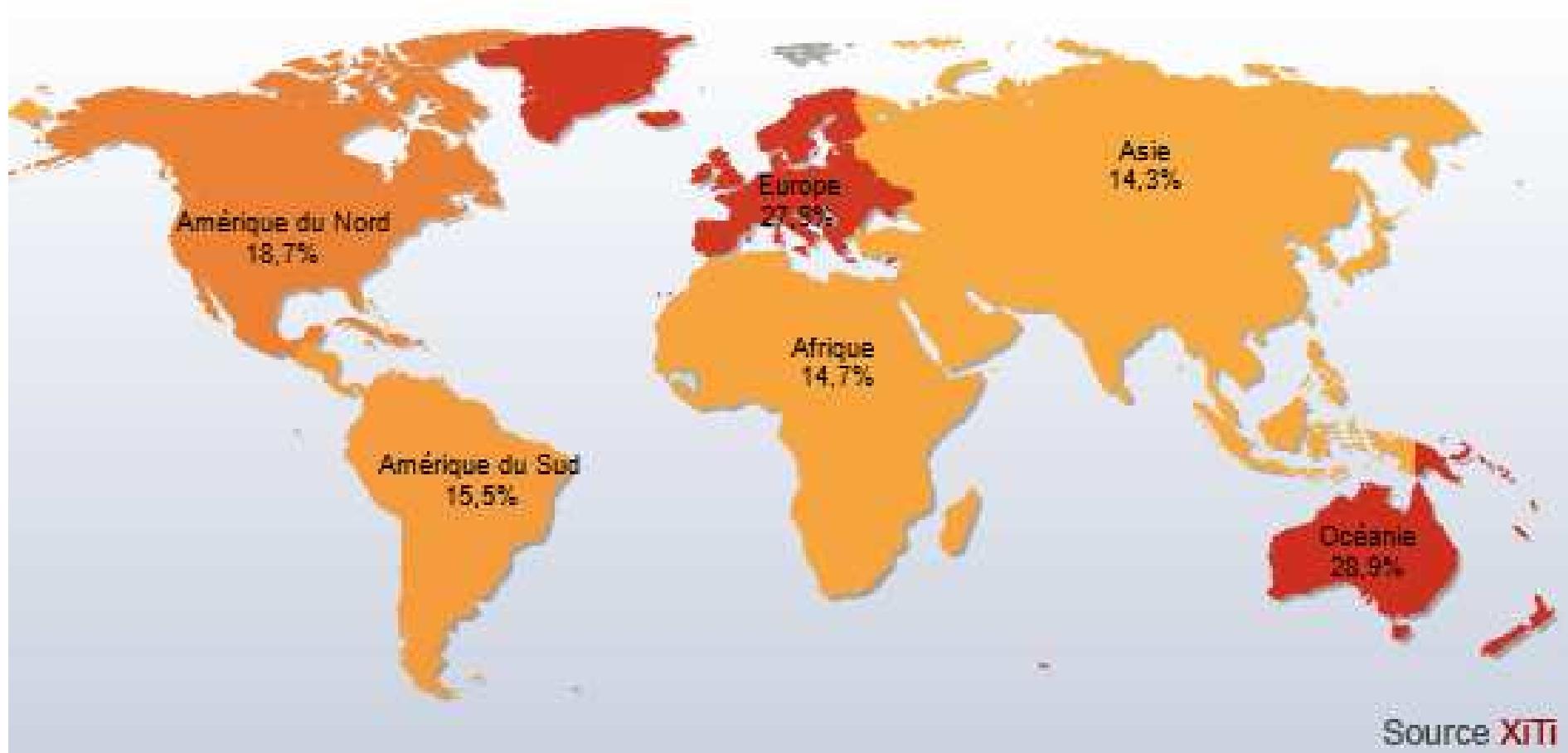
**reduced** costs

This is well understood in developed countries, too : e.g. Germany<sup>b</sup>

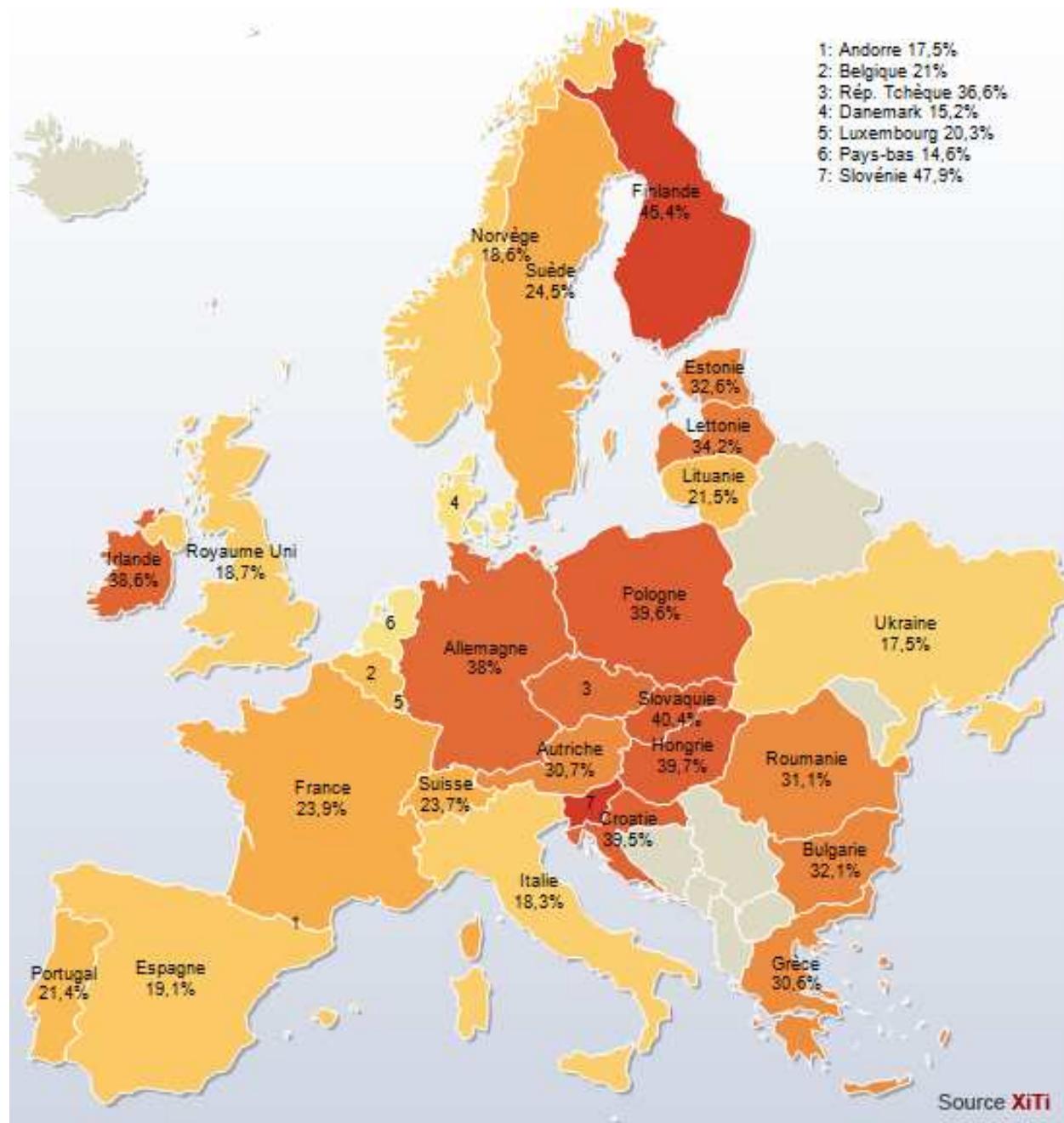
But ...

## Firefox (source : Xitimonitor 07/08)

---



## Firefox (source : Xitimonitor 07/08)



NB : Malte (20,8%) et Monaco (15,6%) n'apparaissent pas sur la carte.

## more examples...

---

France :

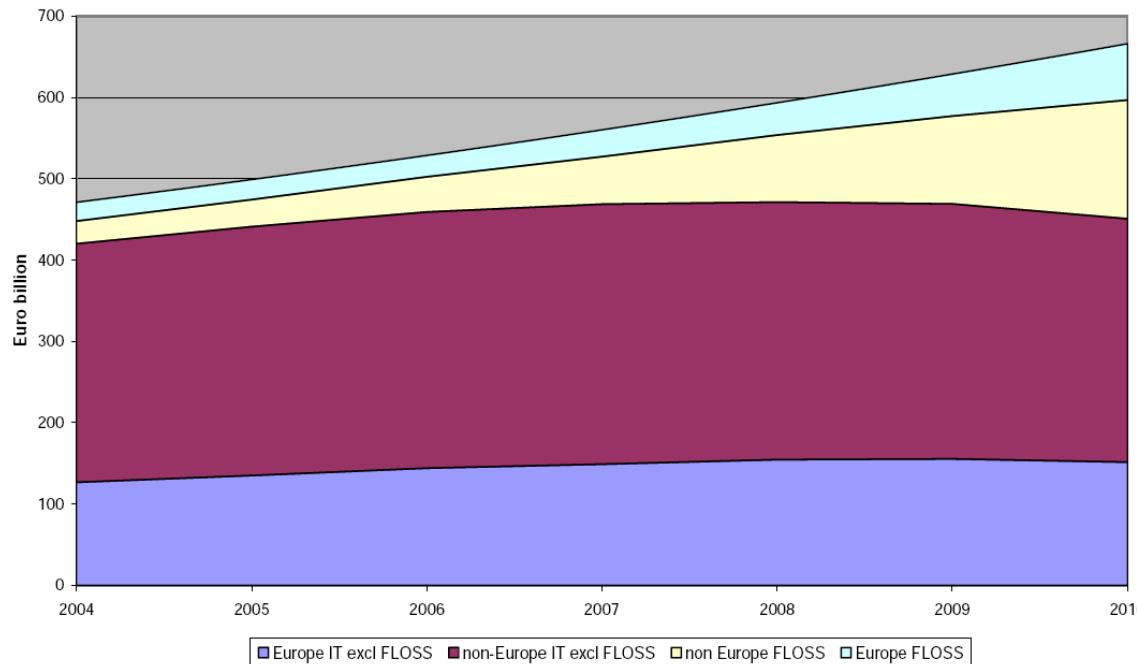
- ▶ <http://impots.gouv.fr> :
  - ▷  $5,5 \times 10^6$  déclaraciones en 2006
  - ▷ fully free software infrastructure
  
- ▶ OpenOffice.org :
  - ▷ Minefi (80.000),
  - ▷ Gendarmerie (70.000),
  - ▷ Intérieur (50.000),
  - ▷ Equipement (55.000),
  - ▷ Douanes (16.000)

*Argentina ?*

## FLOSS study (november 2006) : 32% IT Services, 4% GNP in 2010

---

**Figure 47: FLOSS-related and IT services revenue, Europe and world**

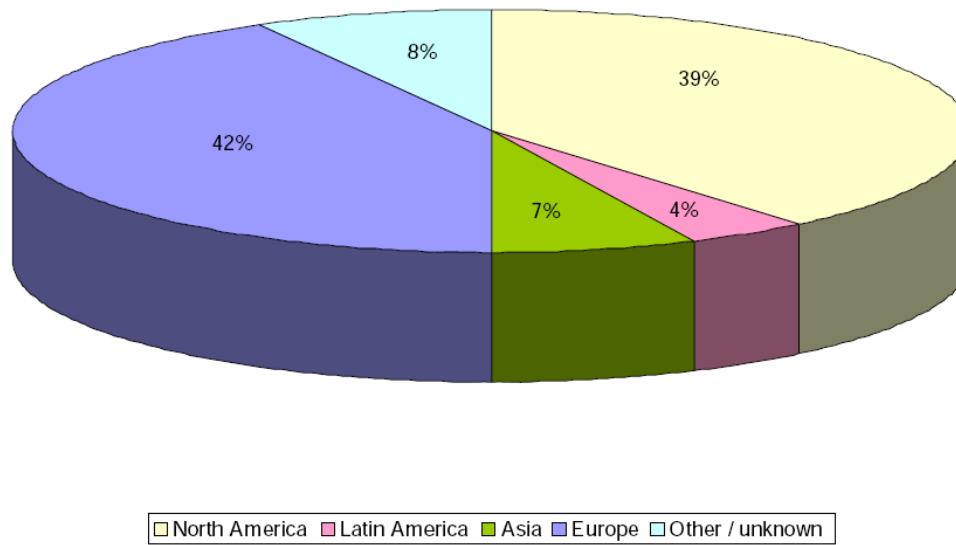


Copyright © 2006 MERIT. MERIT estimates and projections based on sources including Gartner (IT services market size); IDC (Linux server and PC sales); GGDC (software investment ratios). Shares add up to the total (Euro 667 billion in 2010).

## FLOSS study (november 2006) : Europe is fueling the movement

---

Figure 18: Geographical distribution of developers on Sourceforge.



Latin America's share is *dangerously low*. That's not a good idea !

## Some software we use is getting huge ...

---

```
linux-2.6.16.20> slccount .
```

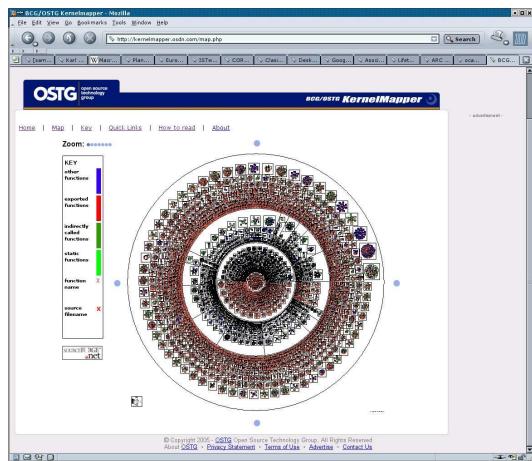
```
[ ... ]
```

```
Totals grouped by language (dominant language first):
```

ansic:	4608272 (95.46%)
asm:	204701 (4.24%)
perl:	5614 (0.12%)
yacc:	2606 (0.05%)
sh:	2230 (0.05%)
cpp:	1769 (0.04%)
lex:	1510 (0.03%)
lisp:	218 (0.00%)
python:	167 (0.00%)
awk:	99 (0.00%)
pascal:	41 (0.00%)

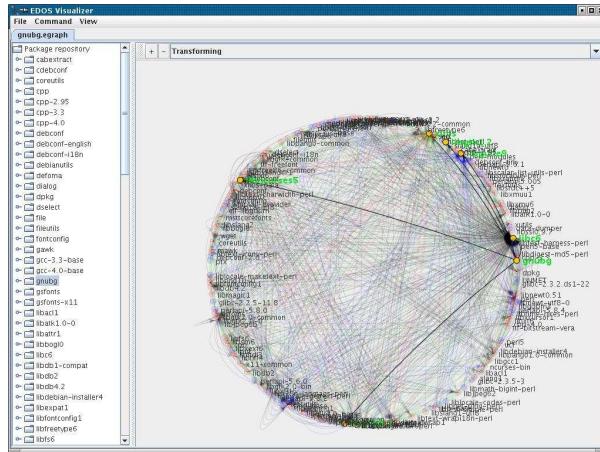
```
Total Physical Source Lines of Code (SLOC) = 4,827,227  
Data generated using David A. Wheeler's 'SLOCCount'.
```

**and quite complex...**



## The software ecosystem is getting complex too...

The relationships among software components are growing intricate...



## A good engineer has a demanding life

---

- ▶ design real-world systems that will go into production
- ▶ understand complex software,  
*at least as much as necessary to modify and adapt it*
- ▶ build complex systems by reusing existing components
- ▶ interact with other, often strongly opinioned, developers

## Yet, we still teach computer science like 20 years ago !

---

- ▶ one algorithm at a time
- ▶ one monolithic program (big or small) for each project
- ▶ one student at a time

this needs to change, and free software is *the key*

## Quality of education and training

Free software = freedom to use, copy and distribute :

- ▶ **level playing field** : *all* students have access to the whole set of tools<sup>a</sup> ; no restrictions, no specific agreements

Well, but in Latin America, proprietary software is free of charge<sup>b</sup> ! **Think twice !**

[...] Gates shed some light on his own hard-nosed business philosophy.

”Although about 3 million computers get sold every year in China, but people don’t pay for the software,” he said. ”Someday they will, though. As long as they are going to steal it, *we want them to steal ours*. They’ll get sort of addicted, and then we’ll somehow figure out how to collect sometime in the next decade<sup>c</sup>.”[...]

Entrevista de Corey Grice, Sandeep Junnarkar (CNET July 2, 1998)

*Timeo Danaos et dona ferentes Virgilio, Eneide, l.2,v.49*

---

<sup>a</sup>Firefox, ThunderbirdOpenOfficeScilab, R,Ocaml, TeXmacs, Eclipse ...

<sup>b</sup>commercial offers, or illegal copy

<sup>c</sup>Esa, es la “proxima década”

## Quality of education and training

---

das Wesen der Mathematik liegt gerade in ihrer Freiheit<sup>a</sup>

Georg Cantor

Free software = freedom to read, understand, modify and distribute the source code :

- ▶ access to a better education (to computer science) :  
no barriers to knowledge  
*“as if an civil engineering student could take part in the building of the Aquitaine bridge”*
  
- ▶ specific curricula :  
several Masters en Free Software are being taught

Free Software is *essential* for education

---

<sup>a</sup>la esencia de la matematica reside en su libertad

## An example : teaching algorithms in a modern way

---

Let's take one of the favorite introductions to dynamic programming

### *Longest Common Subsequence (LCS)*

given two sequences  $X = (x_1, x_2, \dots, x_n)$  and  $Y = (y_1, y_2, \dots, y_m)$ , we wish to find a maximum length common subsequence of X and Y.

For example, for  $X = \text{BDCABA}$  and  $Y = \text{ABCBDAB}$ , the sequence BCBA is such a common subsequence (BDAB is another one).

How do we find one ?

Should we enumerate all subsequences of X and Y, then find the common ones and pick a longest one ?

Hey, that would require exponential time !

## The algorithmic insight, 1

---

We remark that the LCS problem has an *optimal substructure* property :  
for  $X = (x_1, x_2, \dots, x_n)$  and  $Y = (y_1, y_2, \dots, y_m)$ , and  $Z = z_1, \dots, z_k$  an LCS

- ▶ if  $x_n = y_m$  then  $z_k = x_n = y_m$  and  $Z_{k-1}$  is an LCS of  $X_{n-1}$  and  $Y_{m-1}$
- ▶ if  $x_n \neq y_m$  then  $z_k \neq x_n$  implies  $Z$  is an LCS of  $X_{n-1}$  and  $Y$
- ▶ if  $x_n \neq y_m$  then  $z_k \neq y_m$  implies  $Z$  is an LCS of  $X$  and  $Y_{m-1}$

## The algorithmic insight, 2

---

So we can fill an  $n$  by  $m$  table  $c[i, j]$  containing the length of the LCS of  $X_i$  and  $Y_j$

$$c[i, j] = \begin{cases} 0 & i = 0 \text{ or } j = 0 \\ c[i - 1, j - 1] + 1 & x_i > 0, y_j > 0, x_i = y_j \\ \max(c[i, j - 1], c[i - 1, j]) & x_i > 0, y_j > 0, x_i \neq y_j \end{cases}$$

## The algorithmic insight, 3

---

This can be done bottom up with the simple code that follows

```
for i = 1 to n do c[i,0] = 0
for j = 1 to m do c[0,j] = 0
for i = 1 to n do
    for j = 1 to m do
        if x[i]=y[j] then c[i,j] = c[i-1,j-1] +1
                           else c[i,j] = max(c[i,j-1], c[i-1,j])
```

Notice that :

- ▶ we can actually recover an LCS from the matrix  $c$
- ▶ the algorithm runs in  $O(mn)$  time
- ▶ the algorithm requires  $O(mn)$  space

## The algorithmic insight, 4

---

Many lecturers conclude “this is how the `diff` program works !”

*really ?*

Is  $O(nm)$  an acceptable space and time complexity, *in practice* ?

Is `diff` *really* building an  $n$  by  $m$  array of *text lines* ?

Is `diff` *really* comparing *text lines* ?

Is *the student* asking himself these fundamental questions ?

With proprietary software, you would never know.....

With *free software*<sup>a</sup>, things change radically !

---

<sup>a</sup>4 rights :- execute the code- study and adapt the (source) code- distribute the code- distribute the (modified) sources

## A look at diff internals

---

```
apt-get source diffutils  
cd diffutils-2.8.1/src  
less analyze.c
```

...

```
/* The basic algorithm is described in:
```

"An O(ND) Difference Algorithm and its Variations<sup>a</sup>", Eugenio Costantini, Journal of Computer and System Sciences, Volume 33, Number 2, August 1986, pp. 251-266;  
see especially section 4.2, which describes the variations.  
Unless the --minimal option is specified, this code uses a heuristic, by Paul Eggert, to limit the cost to O(N\*\*1.5) at the price of producing suboptimal output for large inputs with many differences.

The basic algorithm was independently discovered as described in  
"Algorithms for Approximate String Matching", E. Ukkonen

---

<sup>a</sup>aha!

Information and Control Vol. 64, 1985, pp. 100-118. \*/

## A look at diff internals, 2

---

```
less io.c
```

```
...
```

```
/* Lines are put into equivalence classes of lines that match.
   Each equivalence class is represented by one of these structures,
   but only while the classes are being computed.
   Afterward, each class is represented by a number.a */
```

```
struct equivclass
{
    linb next;           /* Next item in this bucket. */
    hash_value hash;       /* Hash of lines in this class. */
    char const *line;      /* A line that fits this class. */
    size_t length;         /* That line's length, not counting
                           * trailing whitespace. */
};
```

---

<sup>a</sup>aha!

<sup>b</sup>integer holding a pointer

```
/* Hash-table: array of buckets, each being a chain of equi  
static lin *buckets;
```

## A look at diff internals, 3

---

```
less analyze.c
```

```
...
```

```
/* Discard lines from one file that have no matches in the
```

A line which is discarded will not be considered by the comparison algorithm; it will be as if that line were not there. The file's 'realindexes' table maps virtual line numbers (which don't count the discarded lines) into real line numbers. This is how the actual comparison algorithm produces results that are comprehensible when the discarded lines are counted.

When we discard a line, we also mark it as a deletion or insertion so that it will be printed in the output. \*/

---

```
static void
```

<sup>a</sup>aha!

discard\_confusing\_lines (struct file\_data filevec[ ] )

## Free software makes a difference

---

By looking at the *free source code* of a real-world, industry-strength implementation of the `diff` algorithm, our students have learned :

- ▶ a real-world program is much more than just *one* algorithm
  - ▷ optimize the common case (the  $O(DN)$ )
  - ▷ use hashing where appropriate (line equivalence classes)
  - ▷ reduce the size of the problem (remove lines that are not common)
- ▶ follow references to *freely accessible*<sup>a</sup> research papers
- ▶ documentation, and comments, are essential to understand the code

---

<sup>a</sup>this is really essential !

## Free software and sustainable development

---

**New profiles for our students** Leveraging free software, we can identify three different levels :

**FLOSS technician** : knows how to use some successful FLOSS projects, off the shelf (Apache, Linux, etc.) This can be done with e-learning, at a 2 or 3 year graduate level, see <http://www.eof.eu.org/>.

**FLOSS engineer** : she also knows how to inspect, modify and fix some complex code coming from the FLOSS community

**FLOSS core developer** : she is able to become part of a FLOSS community, and have her changes accepted back in it

See also Paterson's letter in Communication of the ACM (03/06) : "Computer Science Education in the 21st Century".

This is key to nonvolatile, highly qualified jobs.

## Free software also poses novel challenges

---

The challenge :

Manage the complexity of very large software systems, like those in a free software distribution

A difficult problem

- ▶ no single architect
- ▶ version change all the time
- ▶ components (units, packages) come and go

This is why Free Software has created the role of a *distribution editor*

## The role of a distribution editor is *novel* :

**upstream tracking** : must follow the evolution of the sources

*the developer is almost never the packager !*

**integration** : must offer a coherent<sup>a</sup> collection of packages

Coherence relies on properly handling, and checking, *dependencies*

**testing** : metadata will never be complete, so testing is necessary

**distribution** : new packages must be delivered fast, without breaking existing configurations

This is *not* easy :

Mandrake's 6-month release cycle required *30 man-years*.

## An overview of N

---

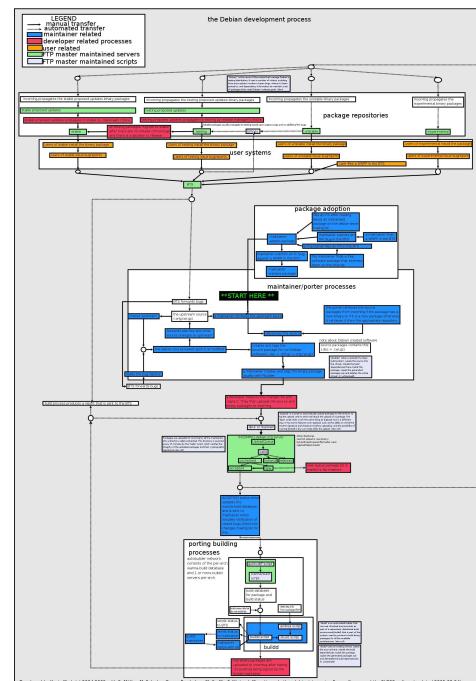


### Legend

- Human Actors
- Automates
- View Points

# An overview of Debians's lifecycle ( $\approx$ 19.000 units)

---



## The EDOS project

---

Funded by the European Community, IST.

Goal : improve the production process of a complex software system, like a free software distribution, using *formal methods* :

- ▶ package management : upstream tracking, dependency checking<sup>a</sup>, thinning, rebuilding from scratch
- ▶ testing
- ▶ distribution : specialised algorithms for P2P clustering and event notification
- ▶ process measurement

This is *radically new* w.r.t. the proprietary software world.

## Metadata in common *binary* package formats is complex

---

**dependencies** package A needs another package B to work properly.

**conflicts** package A that cannot be installed when package B is.

**versioned dependencies and conflicts** dependencies or conflicts can mention package versions.

**complex boolean dependencies** package A can depend on package B AND (package C OR package D).

## An example

---

```
Package : binutils
Priority : standard
Section : devel
Installed-Size : 5976
Maintainer : James Troup <james@nocrew.org>
Architecture : i386
Version : 2.15-6
Provides : elf-binutils
Depends : libc6 (>= 2.3.2.ds1-21)
Suggests : binutils-doc (= 2.15-6)
Conflicts : gas, elf-binutils, modutils (<< 2.4.19-1)
Filename : pool/main/b/binutils/binutils_2.15-6_i386.deb
Size : 2221396
MD5sum : e76056eb0d6a0f14bc267bd7d0f628a5
Description : The GNU assembler, linker and binary utiliti...
The programs in this package are used to assemble, link and...
manipulate
binary and object files. They may be used in conjunction wi...
```

## Checking packagewise installability

---

### The package installation problem

“given a repository R, can I install a package P =(u,v) ?”

Solving this problem is central to :

- ▶ analyse a repository
- ▶ allow distribution maintainers to discover early problems due to the changes in the package versions

## Package installation as boolean constraint solving

---

- ▶ Debian uses unary constraints
  - ▷  $u$  meaning “any version of unit  $u$ ”<sup>a</sup>
  - ▷  $u \ op \ const$  with  $op$  being  $=, >>, <<, >=, =<$  meaning “any version  $v$  of unit  $u$  such that  $v \ op \ const$  is true”.

these can be encoded as boolean constraints : a repository becomes the conjunction of the dependency and conflict relations

- ▶ for Debian repositories, we need also to model the fact that only one version of a unit  $u$  can be installed at a time :

$$\bigwedge_{\substack{v_1, v_2 \in R_u \\ v_1 \neq v_2}} \neg(I_u^{v_1} \wedge I_u^{v_2})$$

## Installation as boolean constraint solving : an example

---

Package : libc6

Version : 2.2.5-11.8

 $\neg(\text{libc6}_{2.3.2.ds1-22} \wedge \text{libc6}_{2.2.5-11.8})$ 

$\wedge$

Package : libc6

Version : 2.3.5-3

 $\neg(\text{libc6}_{2.3.2.ds1-22} \wedge \text{libc6}_{2.3.5-3})$ 

$\wedge$

Package : libc6

Version : 2.3.2.ds1-22

 $\neg(\text{libc6}_{2.3.5-3} \wedge \text{libc6}_{2.2.5-11.8})$ 

$\wedge$

Depends : libdb1-compat

 $\neg(\text{libdb1-compat}_{2.1.3-7} \wedge \text{libdb1-compat}_{2.1.3-8})$ 

becomes

$\wedge$

Package : libdb1-compat

Version : 2.1.3-8

Depends : libc6 ( $\geq$

2.3.5-1)

 $\text{libc6}_{2.3.2.ds1-22} \rightarrow$  $(\text{libdb1-compat}_{2.1.3-7} \vee \text{libdb1-compat}_{2.1.3-8})$ 

$\wedge$

Package : libdb1-compat

Version : 2.1.3-7

Depends : libc6 ( $\geq$

2.2.5-13)

 $\text{libdb1-compat}_{2.1.3-7} \rightarrow$  $(\text{libc6}_{2.3.2.ds1-22} \vee \text{libc6}_{2.3.5-3})$ 

$\wedge$

 $\text{libdb1-compat}_{2.1.3-8} \rightarrow \text{libc6}_{2.3.5-3}$

## Installation as boolean constraint solving : end

---

Now, checking whether a particular version  $v$  of a unit  $u$  is installable boils down to finding a boolean assignment that makes  $v_u$  true, and satisfies the encoding of the repository.

## Installation as boolean constraint solving : end

---

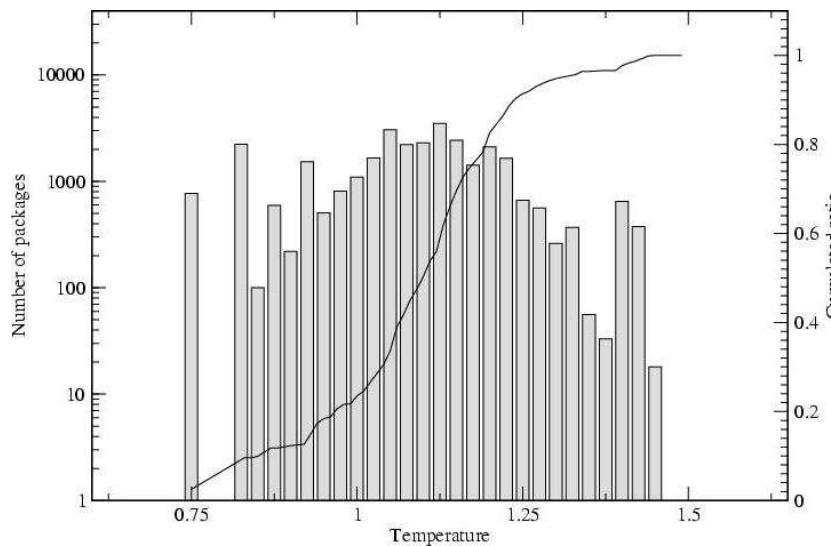
In our example, to test installability of libc6 version 2.3.2.ds1-22 we get the *equivalent<sup>a</sup>* SAT problem

```
libc62.3.2.ds1-22
∧
¬(libc62.3.2.ds1-22 ∧ libc62.2.5-11.8)                                p cnf 5 8
∧
¬(libc62.3.2.ds1-22 ∧ libc62.3.5-3)                                     4 0
∧
¬(libc62.3.5-3 ∧ libc62.2.5-11.8)                                     1 2 -4 0
∧
¬(libdb1-compat2.1.3-7 ∧ libdb1-compat2.1.3-8)                         -4 -5 0
∧
libc62.3.2.ds1-22 →                                                 i.e.b -3 -5 0
(libdb1-compat2.1.3-7 ∨ libdb1-compat2.1.3-8)                           -3 -4 0
∧
libdb1-compat2.1.3-7 →                                         1 3 4 0
(libc62.3.2.ds1-22 ∨ libc62.3.5-3)                                     -1 -2 0
∧
```

## Practical results

---

- ▶ The resulting formulas can be large (median formula size 400 literals) ; luckily, their SAT-temperature is low.



- ▶ Some formulas can be harder<sup>a</sup>.
- ▶ A serious SAT-solver is required.

This is incorporated in the EDOS *debcheck/rpmcheck* tool.

## Installation is NP-complete !

---

We can reduce 3SAT to the Debian package installation problem.

In practice, analyzing the full Debian pool on this laptop ( $\approx 40000$  packages) takes less than 2 minutes.

## Free software as a source for research

The free software community can provide interesting new research problems to computer scientists, and computer scientists can help free software.

Please look at <http://www.edos-project.org>, especially

- ▶ the WP2 deliverable 2.2
- ▶ the subversion repository

<http://www.edos-project.org/xwiki/bin/Main/EdosSvn>

## The last frontier : educating the e-citizen

---

All this is surely nice, but . . . can we stop here ?

IT is becoming pervasive :

- ▶ e-government
- ▶ e-whatever (health, law, tax, etc.)
- ▶ e-vote !

Is it just enough to teach our fellows about our beloved technology ?

Even with free software everywhere ?

Let's make a test . . .

## E-vote

We go for a tour in France... they have some cool technology in store for us...

Do you buy this ?

## E-voting properties

---

**voter verification** only legitimate voters can cast a vote, only once, and only for themselves

**anonymity** nobody knows *somebody else's* vote

**control** the voter can verify that *his* vote is rightly counted

**no coercion** nobody can “prove” having cast a particular vote

Notice that the last 2 requirements seem contradictory...

Rebecca Mercuri proposed a *solution* years ago...

but Italians have shown how to cheat anyway !

## Building solid mental models of computing

---

If we want our students to become educated e-citizens, we face the challenge of transmitting them mental models that make some facts evident to them :

- ▶ computers *execute* instructions
- ▶ instructions *can* be modified
- ▶ computers manipulate *information*
- ▶ we (humans) only have access to a *representation* of information
- ▶ a *representation* of an object *is not* the object !
- ▶ hence, we should never stop questioning technology...

## Conclusion

---

- ▶ free software is key to sustainable development
- ▶ free software, together with open access to research articles, are the key to a better education of computer scientists
- ▶ free software is fueling interesting research on complex systems
- ▶ and yet, our most basic task is to educate the *e-citizen*, not just the computer scientist or the engineer
- ▶ we need to devise new ways of transmitting *knowledge* about computing systems
- ▶ the italian philosopher Vico (circa 1700) has a suggestion :

*conoscere è saper fare*

Thank you for your attention

## Example usage of the EDOS tools

We are playing backgammon but we are unsatisfied by the game performance. We want to know if there is a package with better gameplay. We do a search using `ara.edos-project.org`:

```
backgammon AND (better OR improved OR AI OR intelligent)  
AND section=games
```

Package	Version	...	Popularity	Source	Description
gnubg-bearoffs	0.12-4	...	0.008	gnubg	Improved play for gnubg (gnu backgammon)

The screenshot shows a web browser window with the URL `http://brion.inria.fr/ara/search?p=false&s=popularity&x=false&q=backgammon+AND+%28better+OR+intelligent%29+AND+section%3Dgames`. The title bar says "Search Debian packages using ara". The main content area has a heading "Search Debian packages using ara" and the EDOS logo. Below it, there's a search bar with the query entered. A message says "You can do full boolean (AND, OR, NOT) field-limited regular-expression queries (see examples) You may also: Browse fields by value, especially the tag field or explore metadata and check installability results (ara).". The search results table has columns: Package, Version, Architecture, Archive, Component, Section, Popularity, Source, and Description. One row is shown: gnubg-bearoffs, 0.12-4, i386, stable, main, games, 0.008, gnubg, Improved play for gnubg (gnu backgammon). At the bottom, there are links for "Browse by field", "Last visitors", "Ara home page", "Ara manual page", "packages.debian.org", "The EDOS project", and "Contact". A footer note at the bottom right provides cache statistics.

## Nobody's perfect

---

Let's install gnubg-bearoffs.

```
% sudo apt-get install gnubg-bearoffs
```

Reading package lists... Done

Building dependency tree... Done

Some packages could not be installed. This may mean that you have requested an impossible situation or if you are using the unstable distribution that some required packages have not yet been created or been moved out of Incoming.

Since you only requested a single operation it is extremely likely that the package is simply not installable and a bug report against that package should be filed.

The following information may help to resolve the situation :

The following packages have unmet dependencies.

gnubg-bearoffs : Depends : gnubg but it is not going to be installed

**E : Broken packages**

An instance of the dreaded **Broken Packages from Hell** phenomenon.

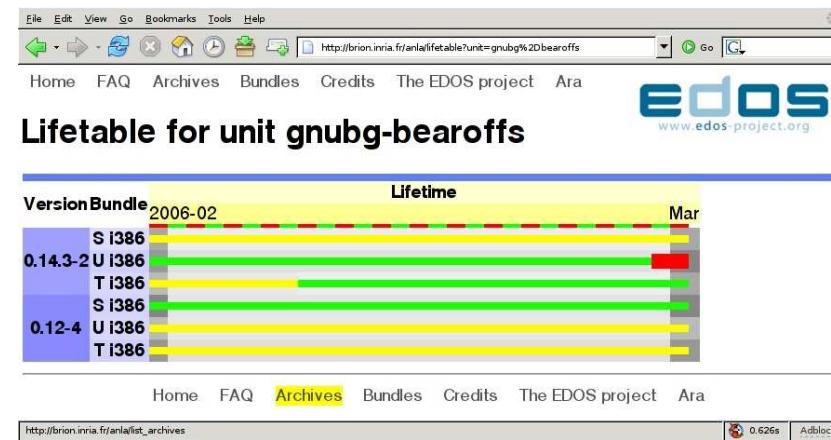
## Why is gnubg-bearoffs not installable ?

---

1. First check Debian's Quality Assurance pages.
2. At <http://packages.qa.debian.org/g/gnubg.html> we see zero reported bugs.
3. At “debcheck” (<http://qa.debian.org/debcheck.php>) : no gnubg-\* package is listed.
4. Searching on gnubg or backgammon on the Debian lists yields nothing.
5. Searching the web yields the maintainers' blog entry stating that it is indeed broken.

## anla, a Debian exploration web interface

- ▶ Integrates our dependency solver `debcheck`. It therefore finds **all dependency and conflict-related installability problems**.
- ▶ Knows of the historical evolution of package metadata.
- ▶ Can explain why packages are uninstallable in a given archive at a given date.



# Diagnosing gnubg-bearoffs

## Output of anla :

The package gnubg-bearoffs 0.14.3-2 is not installable in the bundle U on i386 on 2006-03-01 for the following reasons :

- ▶ gnubg-bearoffs 0.14.3-2 depends on gnubg 0.14.3-3
- ▶ gnubg-bearoffs 0.14.3-2 conflicts with gnubg-data 0.14.3-3
- ▶ gnubg 0.14.3-3 depends on gnubg-data 0.14.3-3

The screenshot shows a web browser window with the following details:

- Address Bar:** http://brion.inria.fr/anla/explain?bundle=U&architecture=i386&package\_i
- Toolbar:** File, Edit, View, Go, Bookmarks, Tools, Help.
- Navigation:** Back, Forward, Stop, Home, Refresh.
- Content Area:**
  - The page title is "Installability".
  - The main text states: "The package [gnubg-bearoffs 0.14.3-2](#) is not installable in the bundle U on i386 on 2006-03-01 for the following reasons:
  - A bulleted list of dependencies and conflicts:
    - [gnubg-bearoffs 0.14.3-2](#) depends on [gnubg 0.14.3-3](#)
    - [gnubg-bearoffs 0.14.3-2](#) conflicts with [gnubg-data 0.14.3-3](#)
    - [gnubg 0.14.3-3](#) depends on [gnubg-data 0.14.3-3](#)
- Footer:** Home, FAQ, Archives, Bundles, Credits, The EDOS project, Ara.
- Bottom Status Bar:** Done, 0.147s, Adblock.