Educating the e-citizen

Roberto Di Cosmo

Université de Paris 7

http://www.dicosmo.org

Bologna, June 26th 2006

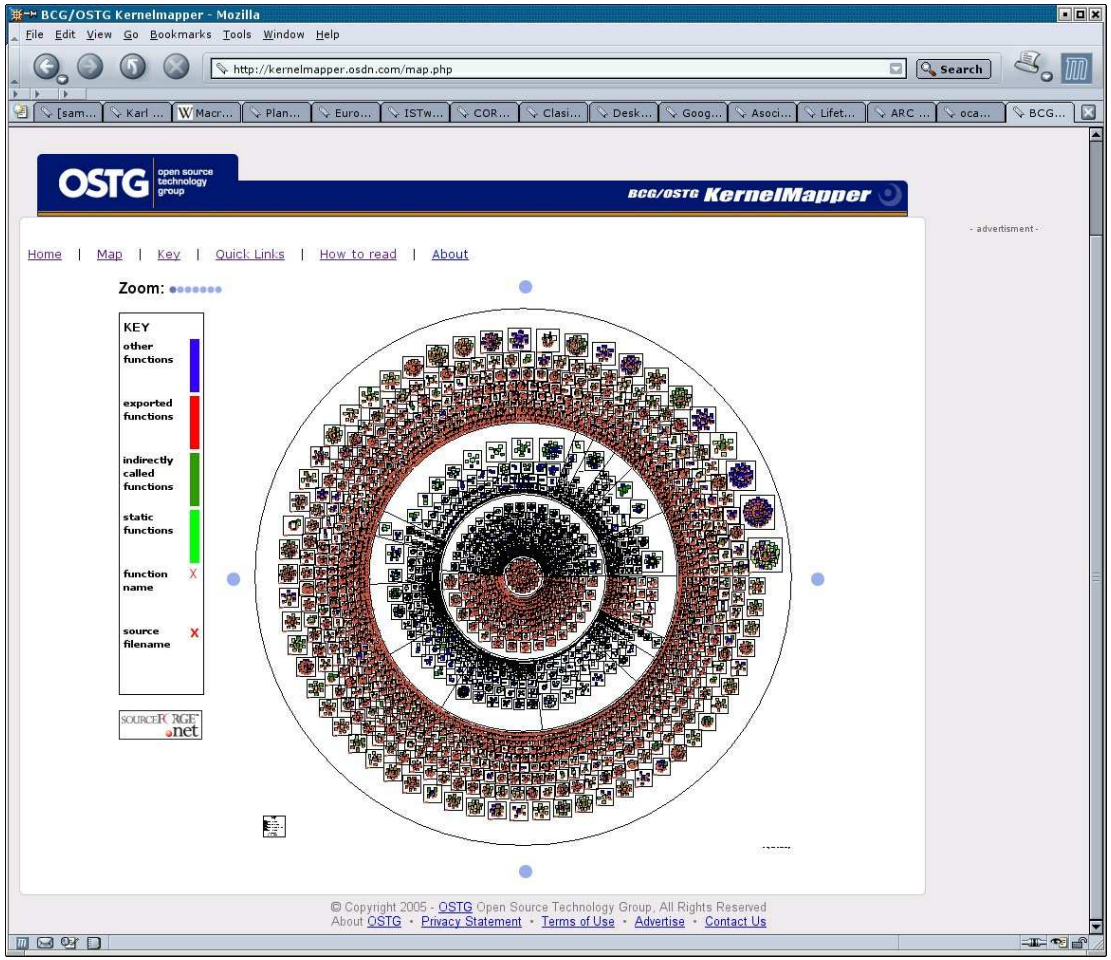
Plan

- Educating the computer scientist
  - the challenge of software complexity
  - the promise of free software (in education, in research)
- Educating the e-citizen

Some software we use is getting huge . . .

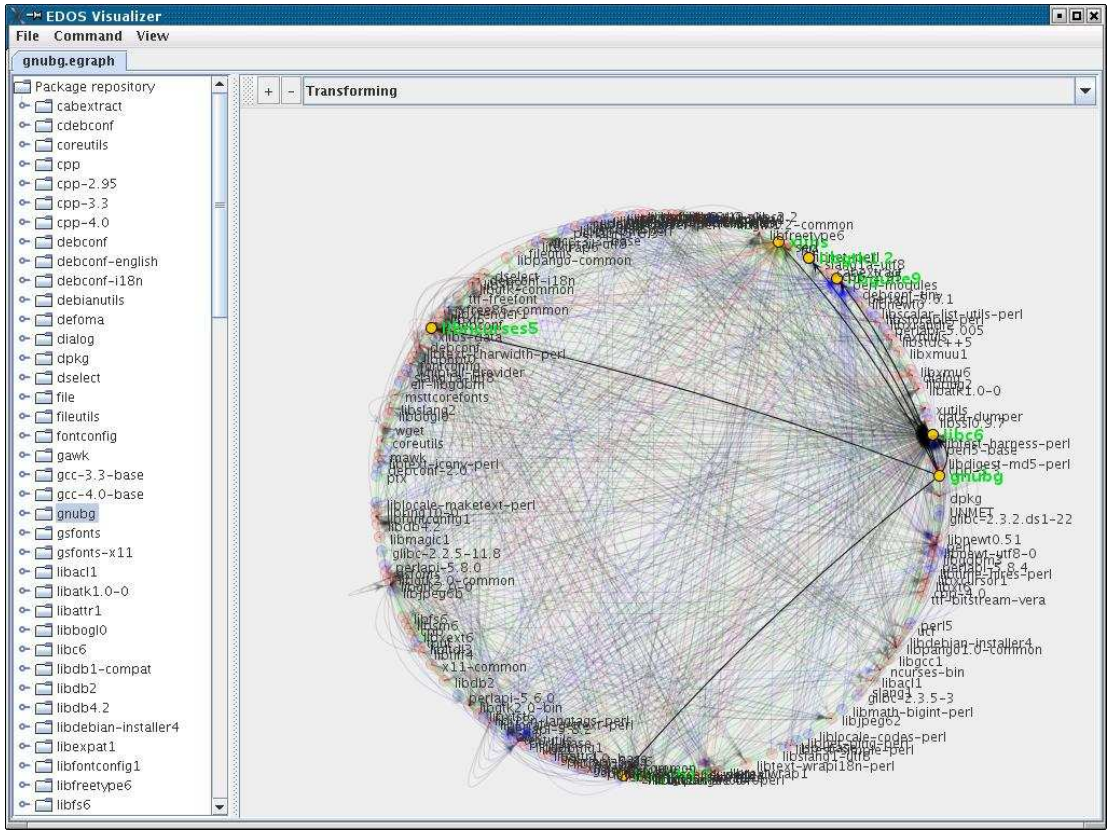
```
linux-2.6.16.20> sloccount .

[...]

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

An example : teaching algorithms in a modern way

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

Longest Common Subsequence (LCS)

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

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

How do we find one ?

Sould 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 $X = (x_1, x_2, \ldots x_n)$ and $Y = (y_1, y_2, \ldots y_m)$, and $Z = z_1, \ldots, z_k$ 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 *your average student* asking himself these fundamental questions ?

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

With *free software*[a], things change radically !

[a] 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", Eugene Myers,
   Algorithmica Vol. 1 No. 2, 1986, pp. 251-266;
   see especially section 4.2, which describes the variation used below.
   Unless the --minimal option is specified, this code uses the TOO_EXPEN
   heuristic, by Paul Eggert, to limit the cost to O(N**1.5 log N)
   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,
   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 in lines_di
   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.  */
struct equivclass
{
  lin[a] 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 its newline.
};

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

[a] integer holding a pointer

A look at diff internals, 3

```
less analyze.c
...
/* Discard lines from one file that have no matches in the other file.

   A line which is discarded will not be considered by the actual
   comparison algorithm; it will be as if that line were not in the file.
   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
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*[a] research papers
- documentation, and comments, are essential to understand the code

[a] this is really essential !

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[a] collection of packages

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

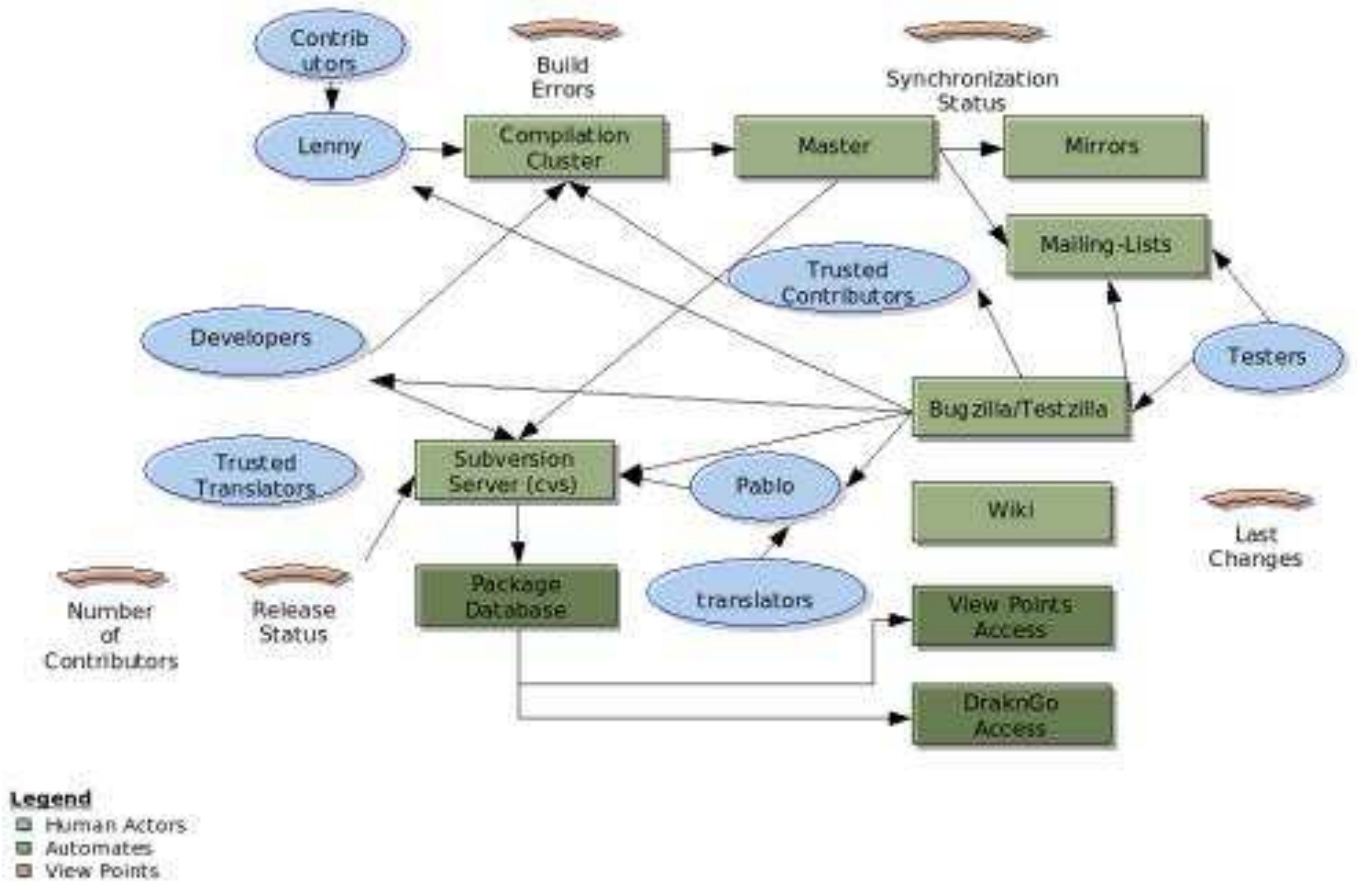
testing : metadata will never be complete, so testing is necessary

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

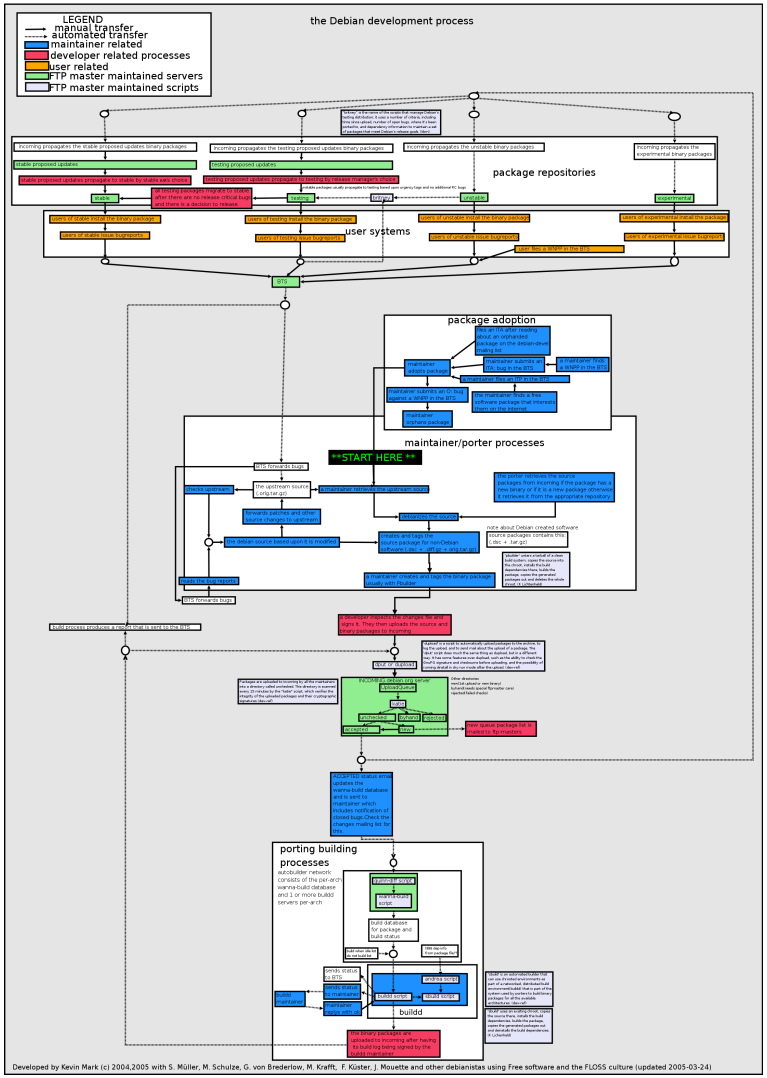
This is *not* easy :

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

An overview of Mandriva's lifecycle (≈ 9.000 units)



An overview of Debians's lifecycle (≈ 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[a], 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.

virtual packages and provides several packages can say they provide a “virtual package” ; other packages can depend on the virtual packages (ex : web browser, mta. . .).

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).

feature dependencies a package can require some other package - any other package - providing feature F (ex : need file /bin/sh).

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 utilities
The programs in this package are used to assemble, link and manipulate
binary and object files. They may be used in conjunction with a compiler
and various libraries to build programs.
```

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”[a]
  - $u\ op\ const$ 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

Package : libc6
Version : 2.3.5-3

Package : libc6
Version : 2.3.2.ds1-22
Depends : libdb1-compat

Package : libdb1-compat
Version : 2.1.3-8
Depends : libc6 (>=
2.3.5-1)

Package : libdb1-compat
Version : 2.1.3-7
Depends : libc6 (>=
2.2.5-13)
```

becomes

$$\neg(\texttt{libc6}_{2.3.2.ds1-22} \wedge \texttt{libc6}_{2.2.5-11.8})$$
$$\wedge$$
$$\neg(\texttt{libc6}_{2.3.2.ds1-22} \wedge \texttt{libc6}_{2.3.5-3})$$
$$\wedge$$
$$\neg(\texttt{libc6}_{2.3.5-3} \wedge \texttt{libc6}_{2.2.5-11.8})$$
$$\wedge$$
$$\neg(\texttt{libdb1-compat}_{2.1.3-7} \wedge \texttt{libdb1-compat}_{2.1.}$$
$$\wedge$$
$$\texttt{libc6}_{2.3.2.ds1-22} \rightarrow$$
$$(\texttt{libdb1-compat}_{2.1.3-7} \vee \texttt{libdb1-compat}_{2.1.3-}$$
$$\wedge$$
$$\texttt{libdb1-compat}_{2.1.3-7} \rightarrow$$
$$(\texttt{libc6}_{2.3.2.ds1-22} \vee \texttt{libc6}_{2.3.5-3})$$
$$\wedge$$
$$\texttt{libdb1-compat}_{2.1.3-8} \rightarrow \texttt{libc6}_{2.3.5-3}$$

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* SAT problem

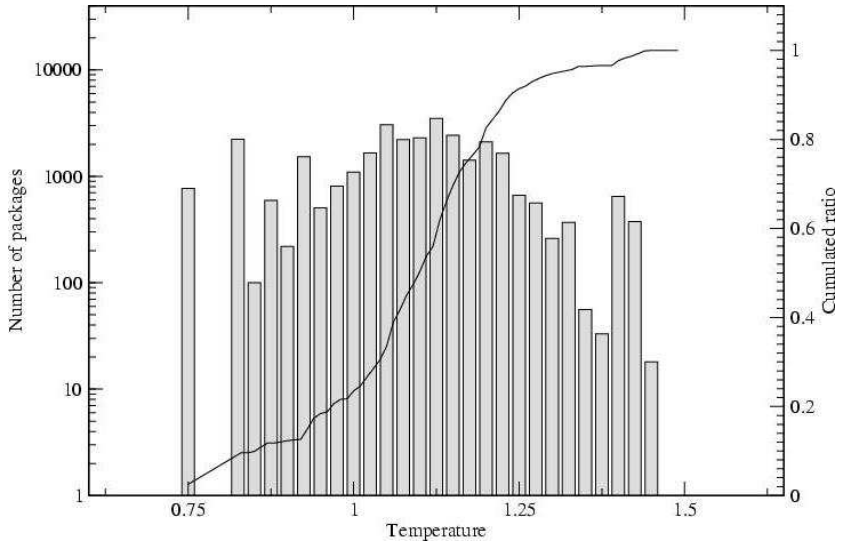
$$
\begin{aligned}
&\texttt{libc6}_{2.3.2.ds1-22} \\
&\wedge \\
&\neg(\texttt{libc6}_{2.3.2.ds1-22} \wedge \texttt{libc6}_{2.2.5-11.8}) \\
&\wedge \\
&\neg(\texttt{libc6}_{2.3.2.ds1-22} \wedge \texttt{libc6}_{2.3.5-3}) \\
&\wedge \\
&\neg(\texttt{libc6}_{2.3.5-3} \wedge \texttt{libc6}_{2.2.5-11.8}) \\
&\wedge \\
&\neg(\texttt{libdb1-compat}_{2.1.3-7} \wedge \texttt{libdb1-compat}_{2.1.3-8}) \\
&\wedge \\
&\texttt{libc6}_{2.3.2.ds1-22} \rightarrow \\
&(\texttt{libdb1-compat}_{2.1.3-7} \vee \texttt{libdb1-compat}_{2.1.3-8}) \\
&\wedge \\
&\texttt{libdb1-compat}_{2.1.3-7} \rightarrow \\
&(\texttt{libc6}_{2.3.2.ds1-22} \vee \texttt{libc6}_{2.3.5-3}) \\
&\wedge
\end{aligned}
$$

i.e.

```
p cnf 5 8
4 0
1 2 -4 0
-4 -5 0
-3 -5 0
-3 -4 0
-2 3 0
-1 3 4 0
-1 -2 0
```

Practical results

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



- Some formulas can be harder[a].
- 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 (≈ 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 the state of Virginia. . . they have some cool technology in store for us. . .

`http://www.alexandriavoter.org/eSlate/eSlate_slide_show.html`

Do you buy this ?

E-voting properties

voter verification only legitmate 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 ! (see the excellent article “Ceci n’est pas une urne” on Andrew Appel’s web page, in english)
- hence, we should never stop questioning technology. . .

Conclusion

Our world is becoming more complex every day :

- 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 an interesting suggestion :

conoscere è saper fare

Thank you for your attention