# Software engineering concepts in ABINIT

I. Software engineering ... for physicists
- "The mythical Man-Month" (1975)
- "No silver bullet" (1986)
- "The Cathedral and the Bazaar" (1997)

II. Assessment of the ABINIT project

III. The future
- More code re-use
- The FSAtom

Ideas to be discussed !

# Software engineering ... for physicists

Our expertise ... is NOT software engineering !

What is software engineering ?

- Not the fact of switching from FORTRAN to C++... !!
- A human science : How to improve the developer's productivity ? ( similarly to machine productivity)
- Potentially very important to us ...
- Compare with hardware evolution : "No single software engineering development will produce an order-of-magnitude improvement in programming productivity within ten years" F. Brooks, *No silver bullet*, 1986.
- The Linux "experience" ...

# The mythical man-month

Ref.: The mythical man-month. Essays on software engineering.
Anniversary edition (1995)
Frederick P. Brooks, Jr. Addison-Wesley

# “The mythical Man-Month” (I)

Essays on software engineering, by F. Brooks

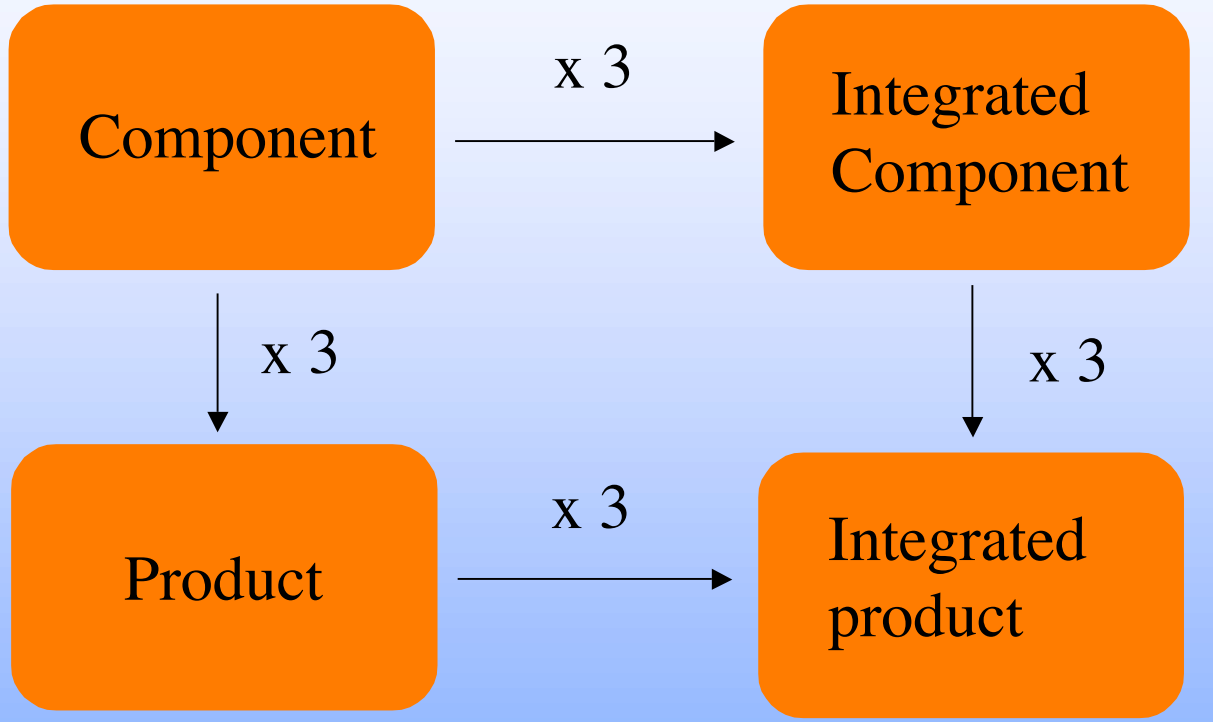
- ✓ First edition 1975, reprinted many times, 20th anniversary edition 1995 (contains “No Silver Bullet”). IBM-360 system chief architect.
- ✓ Basic, easy to read. Some recipes are just organisation recipes.
- ✓ “Large and small, massive or wiry, team after team has become entangled in the tar. No one thing seems to cause the difficulty <...> but the accumulation of simultaneous and interacting factors brings slower and slower motion. Everyone seems to have been surprised by the stickiness of the problem, and it is hard to discern the nature of it. But we must try to understand it if we are to solve it”.

Questions :

- ✓ What really takes time ?
- ✓ How to make a group have better productivity ?
- ✓ Can (software) tools improve the productivity ?

# “The mythical Man-Month” (II)

What takes time ?



# “The mythical Man-Month” (III)

How to make a group have a better productivity ?

(the bearing of a child takes nine months, no matter how many women are assigned)

- “The man-month as a unit for measuring the size of a job is a dangerous and deceptive myth”
- First, each person need training
- Then, software construction is a system effort

  => communication effort can dominate the decrease in individual task time brought by partitioning.
  => need : division of labor + specialisation of function

Also, in our case, each person has his own agenda, his own strengths and weaknesses ... Large productivity variations.

# “The mythical Man-Month” (IV)

Conceptual integrity

- According to Brooks : “Conceptual integrity is the most important consideration in system design. It is better to have a system omit certain anomalous features and improvements, but to reflect one set of design ideas, than to have one that contains many good but independent and uncoordinated ideas.”
- How is conceptual integrity to be achieved ?
- The small team concept (or even the surgical team)
- Solution : Disentangle system architecture and component implementation => centers of decision

“Cathedral building” : a large team and integrity !

# “The mythical Man-Month” (V)

Additional : specificities of software engineering

- Productivity seems constant in terms of elementary statements => programming productivity may be increased as much as five times when the suitable high-language is used (F90 is OK ... for selected parts ...)
- “Representation is the essence of programming” (meaning datastructures, file formats). Not flowcharts ! => towards object-oriented.
- Ways to keep conceptual integrity : Manual - Documentation - Rules
- Self-documentation : the documentation is in the program. Adjustment to humans’ brain limited content !

# “The mythical Man-Month” (VI)

Further ideas :

- A redesign is inevitable, for all components : the only constancy is change itself. So, plan the system for change... And have tools for version maintenance ...
- Program maintenance : unlike for a car, no cleaning, lubricating, repair of deterioration. The needed changes repair design “defects”. These appear because of new functionalities to be implemented. Moreover, fixing a defect has a substantial (20%-50%) chance of introducing another ! Importance of automatic testing.
- Adiabatic changes : quick debugging; references for tests; adding one component at a time.

# No silver bullet

Ref.: “Information processing 1986”, proceedings of the IFIP tenth World Computing Conference, ed. H.-J. Kugler (1986), pp. 1069-76.
Reprinted in The mythical man-month. Essays on software engineering. Anniversary edition (1995)
Frederick P. Brooks, Jr. Addison-Wesley

# “No silver bullet” (1986) by Brooks (I)

Essence / Accidents in software development
(refers to Aristotle categories)

- Essence : the fashioning of the complex conceptual structures that compose the abstract software entity
- Accident : the implementation process itself, actual typing, with hardware and software problems, loss of concentration by the programmer, ...

What is the ratio between them ?

- Brooks argued that (in 1986) essence is more than 10% of development time, that it is inherently complex, and that it is not addressed by emerging software engineering concepts.

# “No silver bullet” (II)

Essence of software development

- construct of interlocking concepts : data sets, relationships among data items, algorithms and invocation of functions ; need specification, design, testing, refinement
- independent of the representation (language)
- “a scaling-up of a software entity is not merely a repetition of the same elements in larger size; it is necessarily an increase in the number of different elements. In most cases, the elements interact with each other in some nonlinear fashion, and the complexity of the whole increases much more than linearly”

# “No silver bullet” (III)

How to address the “essence” bottleneck ?

- Use already existing software ! Software re-use.
  - ✓ Conceptual work already done
  - ✓ Debugged, tested, I/O set-up !
  - ✓ Add “integrated product” to the system (need adequate licence)
  - ✓ Can be completely external (e.g. in our case, ROBODOC)
  - ✓ Can be internal re-use : need modularity !
- Rapid prototyping, then organically grow the software
  - ✓ Iterative extraction of product requirement : the prototype make real the conceptual structure specified, and allow adjusted set-up of “details”
  - ✓ Grow, not build software : incremental development, top-down design. Also psychological: one has something that works.

# “No silver bullet” (IV)

Overall : no silver bullet to kill the werewolf

> Building software takes some uncompressible human time, even if we eliminate the accidental difficulties, and attack the essential difficulties in an efficient way.

This principle was said to be, for software engineering, similar to Heisenberg’s principle, or Gödel’s theorem : a useful information !

Still, the question of group effort occurs ...

# The cathedral and the bazaar

Ref.: http://www.tuxedo.org/~esr/writings/cathedral-bazaar

# “The cathedral and the bazaar” (I)

Eric S. Raymond (1997). Early contributor to GNU.
Analysis of LINUX project. Enthousiastic : read the intro !

Anatomize another open source project. Single out two dozens of propositions related to the process of software development, most of which related to the LINUX experience. Bazaar-like style of development ?!

Again : “software re-use”, “grow, not build”, “rapid prototyping”

LINUX used the GNU General Public Licence. “Free software” or “Open Source Software”. Key concept.

# "The cathedral and the bazaar" (II)

1. Every good work of software starts by scratching a developer's personal itch (Motivation)

5. When you lose interest in a program, your last duty is to hand it off to a competent successor

6. Treating your users as co-developers is your least-hassle route to rapid code improvements and effective debugging

7. Release early. Release often. And listen to your customers.

8. Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone (Linus' law)

Debugging is parallelizable !

# “The cathedral and the bazaar” (III)

9. Smart data structures and dumb code works a lot better than the other way around.

11. The next best thing to having good ideas is recognizing good ideas from others. Sometimes the latter is better.

13. Perfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away.

19. Provided the development coordinator has a medium at least as good as the Internet, and known how to lead without coercion, many heads are inevitably better than one

Cathedral vs bazaar ??  Cathedral AND bazaar !

# Assessment of the ABINIT project

# Assessment of the ABINIT project (I)

Are we doing well ? What's our future ?

Standard measures of success for an atomic-scale simulation tool :

- speed (sequential, parallel) + memory
- set of properties (e.g. : energy, DOS, MD, excited states ...)
- generality (functionals, type of systems, treatment of spin...)

Software engineering measure of success

- user's interface, I/Os
- documentation (for users, for developers)
- stability, portability

Number of users, number of publications

# Assessment of the ABINIT project (II)

(personal view ... biased)

Speed, memory

- For a norm-conserving psp code : OK
- Compared to ultrasoft, PAW, or localised orbitals : bad.
- Parallelism : still missing large-scale, but should come !

Set of properties :

- quite large (the largest ?), steadily improving
- usually not the leader code in one specific domain
- missing : nuclear properties, orbital decompositions, Car-Parrinello, Hartree-Fock ... Also, weak points.

Generality

- quite large (especially spin-orbit, non-collinear)

# Assessment of the ABINIT project (III)

(personal view ... biased)

User's interface, I/Os

- as concern file-oriented : excellent
- need graphical ?
- well-structured files, but need systematic approach (XML, netCDF)

Documentation

- for users : already quite good
- but see Mikami-san's talk

Stability, portability

- stable, but still problems with iterative convergence for some systems (e.g. metal/vacuum ; floppy molecules)
- portability : excellent

# Assessment of the ABINIT project (IV)

(personal view ... biased)

Internal measures of success ... software engineering concepts ...

- GPL licence → rapid building of a user+dev group
  - ✓ no diversion of forces due to management of licences ...
  - ✓ debugging is parallel ... still might be better ...
  - ✓ do we need money from industry ? sponsoring ...
  - ✓ visibility of ABINIT
- Set of rules and protocol to attack the scaling of components to integrated product (portability; documentation; self-documentation;I/Os; interfaces; testing)
- Conceptual integrity : OK, from "Corning", "RESPFN", then "coding standards", but please, try to adhere ! Especially, DOCUMENT your routines ! A challenge...

# Assessment of the ABINIT project (V)

(personal view ... biased)

More on software engineering concepts ...

- Languages :
  - ✓ F90 for compute-related code (recently, evolved towards full use of F90 capabilities , not far of object-oriented)
  - ✓ PERL for automatic testing, and different scripts (+ also shell)
  - ✓ HTML for the doc
  - ✓ XML + Python ?
- Data structures : evolving
- Group work : following Linux
  - ✓ autonomy of different groups
  - ✓ accept all contributions
  - ✓ release early, release often : true, compared to other atomic-scale softwares

# Assessment of the ABINIT project (VI)

(personal view ... biased)

More on software engineering concepts ...

- Re-use : external
  - ✓ ROBODOC : very succesful
  - ✓ MPI/OpenMP, BLAS/LAPACK (a bit weak), make, PERL, CPP
  - ✓ CVS
  - ✓ Sympa, CGI scripts
- Re-use : internal
  - ✓ sometimes OK, but ... see spline routines !
  - ✓ directories ? modules ? lack of documentation ? packaging ?
- Motivation : high if related to the research of each contributor ... but must be convinced that it is worth to follow the rules, and return the module+tests

# Assessment of the ABINIT project (VII)

(personal view ... biased)

## Summary

- Our strengths for the future :
  - ✓ Linux type of organisation, motivated centers
  - ✓ mastering of software engineering concepts
  - ✓ already a wealth of properties
- Major weaknesses (esp. speed) should be overcome soon :
  - ✓ PAW
  - ✓ MPI-parallel FFT

## How to make it pay off for each ABINIT member ??

- Can we live with ABINIT and clear water ?
- For physicists : implementation papers ... feel free ...
- Boost your citation rate !

# A better future :
# More code re-use

# External code re-use

What are the emergent “packages” or “languages” that could play a role in our project ?

- XML : extended markup-language.
  - ✓ Emerging standard for formatted files
  - ✓ Strong structure, readable by humans and machines
  - ✓ Code re-use : tools developed by computer scientists (translator, grammar checker, postprocessor for Web formatting)
  - ✓ In ABINIT v4.0 : input + output (actually CML)
- Python : scripting language
  - ✓ Easy manipulation of complex objects (dynamical, strong data-typing ; object-oriented)
  - ✓ Available interfaces with XML, graphical routines , GUI, ...
- NetCDF

# Internal code re-use

How to improve internal code re-use ?

- More self-documentation ?
- Constitution of more autonomous libraries ?
- Automatic indexing ?

# The Free Software Project for Atomic-Scale Simulations (FSAtom)

# Beyond the licence : links between developers ?

present situation in the Atomic-Scale Computation community

- many different codes coexist, with different licences
- biodiversity is nice, but should allow cross-comparison
- (friendly) rivalry

search for links (easier, thanks to the FS concept)

- CECAM 2001 discussion workshop (7 participants)
- CECAM 2002 ‘Open Source software for microscopic simulations’ (25 participants - from large-scale atomistic, to Quantum Monte Carlo, through ab initio - nearly no Quantum Chemist)

# Toward links ... objects

there are many codes implementing the same concepts (with a loss of human time)

but accurate comparison is not done on a systematic basis

Common objects ?

pseudopotentials, list of atoms, wavefunctions ...

files ? structured arrays ?

Common patterns and methods ?

# Possible actions beyond the licence

standards for file formats

standards for objects and methods

constitution of libraries

protocol for comparison of code accuracy

protocol for comparison of code speed

organization of events for discussion, exchange, learning ...

Need time, and a structure ...

# FSAtom : I. Purpose

The Free Software project for Atomic-Scale Computations has the aim:
- to spread the use of the “free software” concept in the community of Atomic-Scale computation software developers,
- to improve the awareness of modern software engineering concepts, and
- to constitute the natural place for interactions between different groups of developers in this field.

# FSAtom: II. Means of Action

Web site (hosted by CECAM), with mailing lists and links (http://www.fsatom.org)

workgroups to organize the collaboration between developers (file exchange, code testing, definition of objects, exchange of expertise ...)

organisation of workshops and tutorials

contact with the Free Software Foundation

contact with funding agencies

# FSAtom: III. Organisation

Steering committee (elected at CECAM 2002)

- D. Ceperley
- X. Gonze (elect. chair)
- K. Hinsen (resp. workgroup interfaces and middleware)
- K. Jakobsen (resp. workgroup pseudopotentials)
- L. Kale
- M. Marques
- G. Martyna
- D. Van Der Spoel (resp. workgroup testing MD)
- G. Zerah (resp. workgroup testing DFT)

# Summary

Brief history of “software engineering”

- Cathedral AND bazaar

Assessment of the ABINIT project

- Speed, properties, generality
- Linux style group effort !!
- Conceptual integrity : rules, self-documentation ...
- Need more software re-use : external, internal
- XML, Python, NetCDF (also the future within FSAtom)

# Free Software
# or
# Open Source Software

(This is kind of an appendix to the talk, in case of questions)

(See http://www.fsf.org and http://www.fsf.org/licenses/licenses.html )

# A key : “Free software” or “Open Source”

Free for freedom, not price

- user’s freedom 1 : unlimited use for any purpose
- user’s freedom 2 : study and modify for your needs
- user’s freedom 3 : copy
- user’s freedom 4 : distribute modifications

If one freedom is missing : “proprietary software”.

From copyright to freedom (copyleft?)

- copyright allows licensing
- licenses grants freedom

Terminology : Free software=Open source=Libre software

# Licences ...

Many types

- GNU General Public Licence
- GNU Lesser General Public License (links are possible)
- BSD licence
- X11 licence, Perl licence, ...
- public domain release

GNU General Public Licence

- grants four freedoms
- protection of freedom
- «□accination□

# Free software properties

Reasons to use

- stability (“peer review”)
- short turn-around cycles for bugfixes : no loss of prestige
- future-proof
- secure investment

Reasons to develop

- altruism, fun, satisfaction
- testing, education
- strategic reasons: society, democracy, ...