Scientific Software projects: my modest experience with QUANTUM ESPRESSO

P. Giannozzi
Università di Udine and IOM-Democritos, Trieste, Italy

20 March 2013

Quick introduction to QUANTUM ESPRESSO

In the following I am concentrating on *free/open-source scientific software projects*. The project I have been working on, QUANTUM ESPRESSO, (`quantum-espresso.org`) performs electronic-structure calculations from first principles, with applications in condensed-matter physics, chemistry, biology, engineering, materials science.

QUANTUM ESPRESSO is not a very big project, involving

- a hard-to-quantify number of developers: about a few tens;
- an even harder-to-quantify number of users: a few hundreds research groups, $\mathcal{O}(1000)$ individuals subscribed to the mailing list;
- no less than 300K lines of "core" code (mostly Fortran-95, some C).

It is not small either, though!

http://www.quantum-espresso.org

QUANTUMESPRESSO

HOME PROJECT DOWNLOAD RESOURCES PSEUDOPOTENTIALS CONTACTS NEWS & EVENTS

SEARCH

Search here...

Forum

NEWS

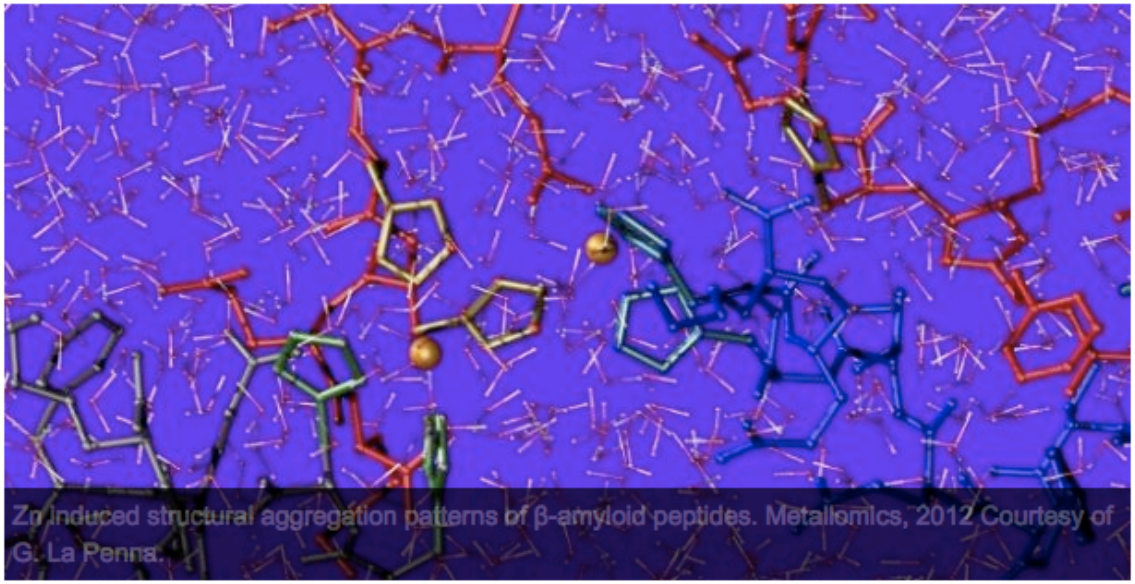
13.12.12

QUANTUM ESPRESSO EVENTS 2013

Hands-on Tutorial on Electronic Structure Computations, ICTP Trieste, 14-18 January.

Workshop on Computer Programming and Advanced Tools for Scientific Research Work & Quantum ESPRESSO Developer Training, ICTP Trieste, 11-28 March

(more events are planned, stay tuned)



Zn induced structural aggregation patterns of β-amyloid peptides. Metallomics, 2012 Courtesy of G. La Penna.

QUANTUM ESPRESSO

is an integrated suite of Open-Source computer codes for electronic-structure calculations and materials modeling at the nanoscale. It is based on density-functional theory, plane waves, and pseudopotentials.

Scientific software vs other software

Free/open source scientific software differs from other kinds of software in several important respects:

- *Software is the mean, not the goal.* Nobody works just for better software; the goal is better science (or better jobs, see later).

- *Little help is coming from outside.* Scientific software is mostly written by scientists working in the field. The role of other people in development is limited, although sometimes very useful.

- *Developers are amateurs programmers.* Scientists can be really awful programmers: bound to old habits, skeptical about new techniques, knowing little to nothing about software engineering.

More sources of trouble in scientific software

- *Most development done by people on short-term positions*. Typically post-docs or graduate students, who have a more pressing task than software (or even science): finding the next job. A lot of effort is spent, and sometimes wasted, in taking over half-baked work.

- *Funding for software development erratic to non-existent*. No way to convince funding agencies that scientific software is an important tool that must be properly developed and maintained: you have to promise flashy things (new sources of energy, new electronic devices, cures for cancer/hemorrhoids/stupidity...), with very few exceptions.

- *Too many people reluctant to contribute*: "I have done something that gives me a competitive edge, why should I contribute it?", or the ever-green "I have no time", are too-frequent afterthoughts.

The ultimate source of trouble in scientific software

Development of really new and important stuff may take years (and this wouldn't be a problem) and it is very often done working on code versions *that are obsolete by the time the development is completed*.

Backporting changes done in an archaic code version to a much more recent version takes a disproportionate amount of time and effort. Sometimes it is done anyway, more often than not it isn't. Eventually the software *forks into many incompatible and incomplete versions* ...

Fighting fragmentation was one of the original motivations to start the PWscf code that later became part of QUANTUM ESPRESSO.

A few ideas (but confused)

- We shouldn't make life too hard for those who contribute: let's avoid Soviet-style central planning and strict programming rules
- Let's introduce methods, tools and techniques that are common in more organized open-source software projects. SCM (Source Configuration Management) software like CVS or SVN allows
  - to keep track of everything that has been done since the beginning
  - to have a unified code base, with "branches" if needed
  - to make "stable" releases so that development can go on almost continuously while still guaranteeing some stability and reliability
- Let's proceed smoothly, with incremental changes, keeping input and output compatibility with previous releases as much as possible.

QE as a distribution

QE is not organized as a monolithic code, but it is rather a *distribution* (integrated suite) of "packages" that can be installed on demand and with varying degrees of integration. It is possible to contribute:

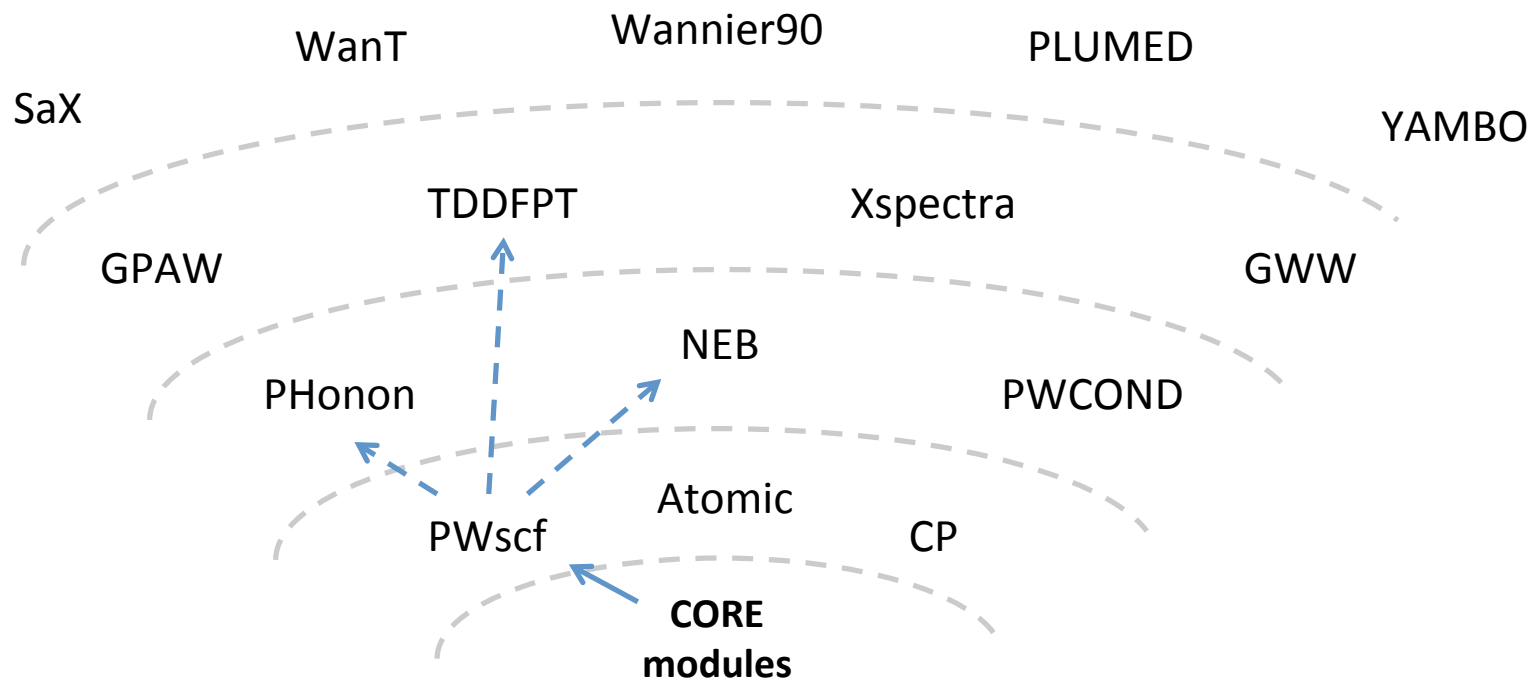
- a small (or large) piece of code to an existing package; or
- a new package that uses QE as a library; or
- a "plugin" that modifies QE, adding a new functionality; or
- a new "external" package that just reads data file produced by QE.

Next slides courtesy of Filippo Spiga.

The development model

- QUANTUM ESPRESSO is **not a monolithic application**, but an integrated ecosystem thriving around a small number of core components developed and maintained by a small number of developers
- the ecosystem is designed so as to be alien-friendly: a number of **third-party QE-compatible applications and add-ons**, often designed to be code-agnostic, are distributed with QE (notable examples include wannier90, yambo, EPW, WanT, XCrysDen, ...)
- the environment that allows the ecosystem to prosper is provided by the QE-FORGE.ORG platform, **freely available** to researchers and developers from all over the world

Quantum ESPRESSO package portfolio



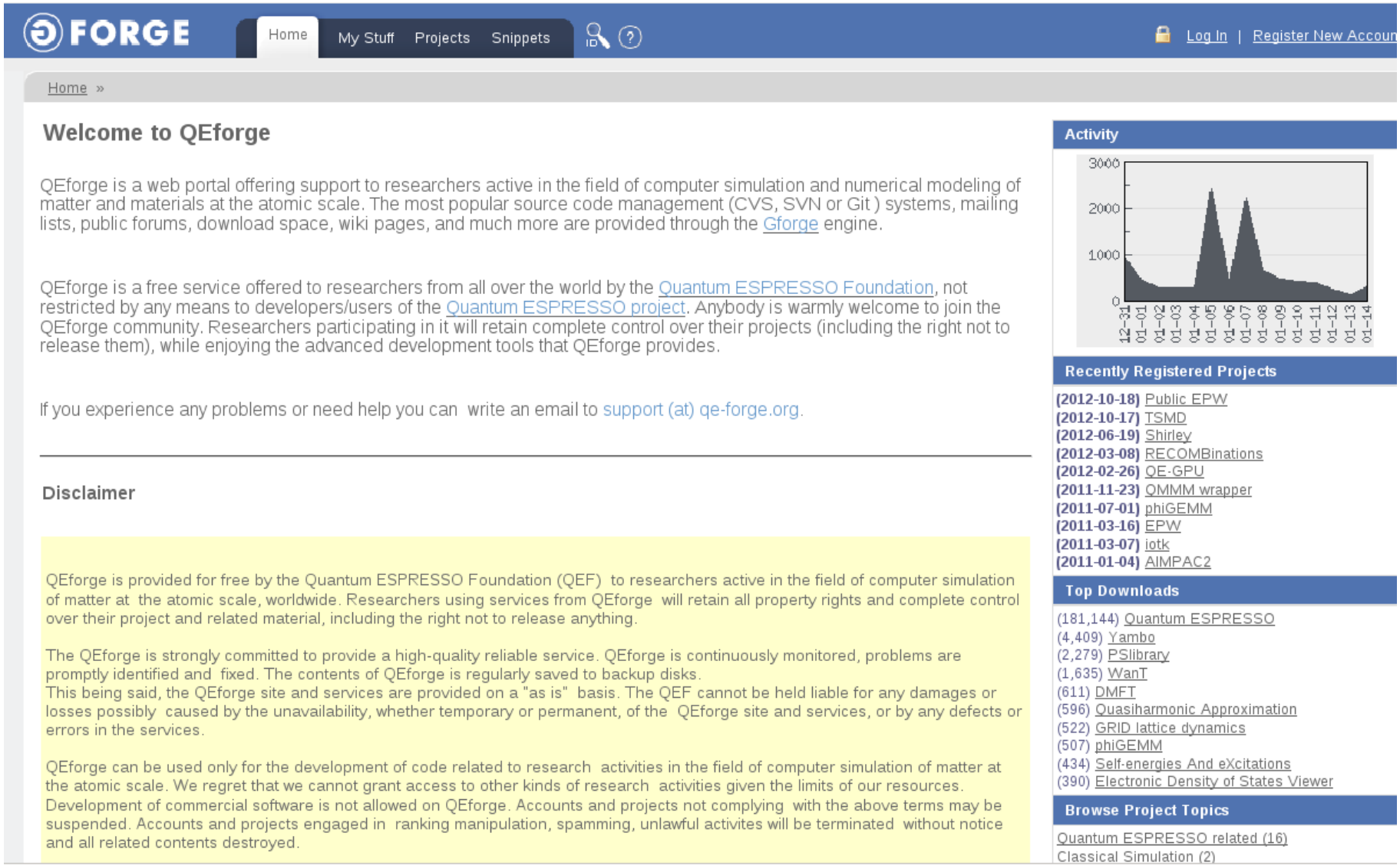
CVS and SVN

CVS (Concurrent Version System) has been the first Source Configuration Management software used in QE (it was a big step forward!). More recently we moved to SVN: more modern and powerful, has interface with `git` and other fancy stuff, but it is still simple enough for non-geeks (e.g. physicists). Current organization:

- *trunk*: development goes on here – open read-only to everybody
- *branches*: major new developments, disruptive changes, very experimental features, things that have a long time before being released (if ever) ... – branches may or may not be public
- *external*: packages that are be developed in a separate SVN trunk can be downloaded into the main QE trunk – access may be restricted to specific (usually expert) developers.

Development environment: QE-forge

Download space, CVS/SVN, mailing lists, forums, bug tracking, ...



Releases

Release early, release often (open-source conventional wisdom). Releases are labelled as $N.M.p$, where N=major, M=minor, p=bugfix.

- *Major*: when something really important changes, e.g.
  1. First public release of PWscf
  2. Conversion from f77 to f90
  3. Merge with the CP and FPMD codes (QUANTUM ESPRESSO)
  4. New XML-based data file format
  5. Major package and directory reorganization

- *Minor*: when some important new functionality is being added

- *Bugfix*: only bug fixes; occasionally, minor new functionalities that don't break any existing one are allowed to sneak into a bugfix release.

Release F.A.Q.'s

- *Who decides when and how a release is done?* Informal discussions on the developers mailing list

- *What happens then?* Development of new stuff is temporarily stopped: nothing new or potentially "dangerous" is added, and all attention is dedicated to fix bugs and to stabilize the distribution

- *Are all packages released at the same time?* Not necessarily: "external" packages can be independently released, as long as there is no compatibility problem; all others are typically released together

Future directions

In order to simplify and make more effective the development, and to reduce "long-range" unintended effects of modifications that affect in a hard to guess other parts of the code, it is planned to

- transform QUANTUM ESPRESSO into a library, with a set of well-defined API's (Application Programming Interface)

or maybe more realistically:

- better document what each routine does, reducing in the process their size and the number of things they do.