Building ONETEP
Scientific software design and development practices

James C. Womack
University of Southampton, UK

Faraday Institution workshop
30 Oct 2018
Preamble

Slides will soon be available at jcwomack.com

References — ask me, or see slides online
Who am I?

Or: Why should I listen to you?

➤ I am a postdoctoral researcher in computational physical sciences.

➤ I develop and implement theoretical models in software.

➤ My focus is modelling matter at the atomic scale using quantum theory.
Who am I?

I have worked on several scientific software projects...
Outline for this talk

Q: How does scientific software development work in practice?

1. Thoughts on scientific software development
   ▶ What defines scientific software?
   ▶ What is “good” scientific software?

2. ONETEP as a research tool
   ▶ The motivating problem
   ▶ What it does and how it does it

3. ONETEP as a software package
   ▶ Distribution, usage and governance
   ▶ Design and supporting infrastructure
   ▶ Software development practices
Scientific software
What defines scientific software?

Some common attributes of scientific software packages:

1. Motivated by needs of developers
   ▶ Developers and users often overlap

2. Not developers’ main “product”
   ▶ A means to perform research

3. Continually developed, extended
   ▶ Always a prototype, never finished
   ▶ Nature of problem changes

These are some general characteristics I have observed: they are not unique to scientific software (e.g. Linux kernel), nor are they universal to all scientific software.
What defines scientific software?

Contrast this with software to control an ATM...

1. Designed for users, not developers
2. Software is developers’ product
3. Clear, fixed specification

Image: “Nice ATM” by Flickr user Jason Cupp, used under CC BY 2.0
Scientific software: some key principles

**Key principles**

- Scientific record
- Multiple developers
- Avoid regression
- Confidence in results
- Validate new functionality
- Help future developers
- Support users

- Simplified debugging
- Code reuse
- Scientific record
- Easier to extend

**Version control**

- Scientific record
- Multiple developers
- Avoid regression
- Confidence in results

**Modularity**

- Scientific record
- Code reuse
- Easier to extend

**Testing**

- Validate new functionality

**Documentation**

- Help future developers
- Support users
Scientific software is scientific infrastructure

A scientific software package is similar to a large shared scientific facility:

1. Designers (of experiments) are users
2. The product is research, not the instrument itself
3. New experiments add new capabilities

Image: Google Maps, Imagery (c)2018 Google, Map data (c)2018 Google
Scientific software is scientific infrastructure

Not just a metaphor... 

- Research tools for scientific communities
- Significant investment of human-time
- Requires maintenance and long-term planning
- Enables applications beyond those possible for any individual researcher or group

e.g. EPSRC’s “Software as Infrastructure” strategy

Image: Google Maps [with logo overlaid], Imagery (c)2018 Google, Map data (c)2018 Google
Scientific software: the good, the bad and the ugly

A. Developer
J. Gen. Sci. 2018

science.py

#?*!

?!

?!

A
B C
D E F

#?*!

?!
Scientific software: the good, the bad and the ugly

D, E, F et al. Nature, 2019
Scientific software: the good, the bad and the ugly

- Be experimental, but then refine and clean up
- Plan development to avoid unmaintainable messes
- Well-written code is easier (and quicker) to extend
ONETEP as a research tool
Quantum chemistry

Time-independent electronic Schrödinger equation

\[ \hat{H}\Psi = E\Psi \]

- Solutions provide information about properties of materials and molecules
- Intractable many-body problem for all but the smallest atomic system

Self-consistent field approach (SCF)

\[ \hat{H}\psi_i = \varepsilon_i \psi_i \]

- Convert $N$-body problem into $N$ more-tractable 1-body problems
- Electrons interact through a mean field which is self-consistently solved for
Density functional theory (DFT)

\[ E[n] = T_s[n] + E_{es}[n] + E_{xc}[n] \]

- A popular and widely adopted SCF method
- Balances accuracy and computational expense
- Conventional DFT: \( O(N^3) \) scaling

The \( O(N^3) \) scaling of conventional DFT rapidly becomes problematic for > 1000 atoms...
Linear-scaling DFT in ONETEP

**Motivation:** Practical quantum chemical methods for modelling large systems

**Premise:** Reformulate DFT to achieve $O(N)$ scaling

- Possible due to “nearsightedness” [10] of electronic interactions
- Locality of interactions manifested in the density matrix, $\rho(r, r')$
- ONETEP expresses DFT as a direct energy minimization wrt $\rho(r, r')$

$$E = \min_{\rho} E[\rho(r, r')]$$

Many interesting theoretical and computational techniques are used to implement this in practice [7, 11].

We will focus on the result. . .
Linear-scaling DFT in action (2005)

64 processors

2606 atoms

Linear-scaling DFT in action (2016)

- Iridis 4
- 512 cores
- 13696 atoms

![Graph showing total calculation time vs. number of atoms for B97M-V, PBE, and PKZB methods.]

Iridis 4
512 cores
13696 atoms
A tool for researchers

ONETEP computes and analyses quantities relevant to physical scientists

An incomplete list:

- Treatment of excited states [12]
- Implicit solvent model [13]
- Polarizable force field embedding [14]
- Energy decomposition analysis [15]
- Electronic transport [16]

www.onetep.org
ONETEP as a software package
## Distribution and licensing

<table>
<thead>
<tr>
<th>License</th>
<th>Cost</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collaborator</td>
<td>-</td>
<td>Individual, specific project</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Includes source</td>
</tr>
<tr>
<td>Academic</td>
<td>£500</td>
<td>Entire research group</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Includes source</td>
</tr>
<tr>
<td>Commercial</td>
<td>Purchase of Materials Studio</td>
<td>Non-academic users</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GUI interface</td>
</tr>
</tbody>
</table>

- All members of the [CCP9](#) community are entitled to collaborator agreements.
- Daussault Systèmes [BIOVIA Materials Studio](#) packages ONETEP into a suite of materials science tools with a single GUI.
Usage

150 academic licenses
(individuals and groups)

20-30 academic collaborations
(PIs and groups)

Daussault Systèmes BIOVIA’s products have
**high market penetration**
in top chemical companies
(pharma, oil, automotive, materials, etc.)
Developers

**ONETEP Developer’s Group (ODG)**

Group of core developers who manage licensing, releases, relationship with BIOVIA:

Jacek Dziedzic,\(^1,2\) Peter D. Haynes,\(^3\) Nicholas D. M. Hine,\(^4\)
Arash A. Mostofi,\(^3\) Mike C. Payne\(^5\) and Chris-Kriton Skylaris\(^1\)

\(^1\) University of Southampton, \(^2\) Gdańsk University of Technology,
\(^3\) Imperial College London, \(^4\) University of Warwick, \(^5\) University of Cambridge

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>30</td>
</tr>
<tr>
<td>code contributors (past and present)</td>
<td>active users of repository (developers and expert users)</td>
</tr>
</tbody>
</table>
Source code

- Written in Fortran (2003 standard), built using GNU Make
- Bundled support utilities written in other languages (e.g. Bash scripts)
- Fortran modules used to encapsulate related procedures and data types
Parallelism

- Designed to run on parallel (super)computers
- Necessary for large-scale calculations
- Combined MPI [17] and OpenMP [18] parallelism.

**MPI** Work divided between processes which communicate by *message passing*

**OpenMP** Processes subdivided into threads, with *shared memory* access
Interaction with other software

Designed for Unix-like systems: Linux, Mac OS X (Windows via WSL)

Supporting software is required to build and run ONETEP:

<table>
<thead>
<tr>
<th>Software/library</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran compiler (F2003, OpenMP)</td>
<td>Intel Fortran, GNU Fortran</td>
</tr>
<tr>
<td>MPI library</td>
<td>Intel MPI, OpenMPI</td>
</tr>
<tr>
<td>Linear algebra libraries</td>
<td>BLAS, LAPACK, ScaLAPACK</td>
</tr>
<tr>
<td>Fast Fourier transform (FFT) library</td>
<td>FFTW, FFTW3</td>
</tr>
<tr>
<td>Multigrid solver library</td>
<td>DL_MG</td>
</tr>
<tr>
<td>Vendor-specific FFT libraries</td>
<td></td>
</tr>
</tbody>
</table>
Hardware

- ONETEP runs well on conventional CPU-based machines
- Scales to 1000s of cores [18]
- More cores and more memory per core are generally advantageous
- GPU-port is a work-in-progress [19]
- Can also run on desktops/laptops
Aside: DL_MG

DL_MG [6] is an open-source Poisson solver library distributed with ONETEP

- Solves several variants of the Poisson equation in real-space:

\[
\nabla \cdot (\varepsilon(r) \nabla \phi(r)) = -4\pi (n(r) + n_{ions}[\phi](r))
\]

- Designed for use in large scale electronic structure calculations
- Multigrid approach [20] scales well with problem size and converges rapidly
- Iterative high-order defect-correction reduces discretization error

Enables open BC electrostatics and implicit solvent functionality in ONETEP
Aside: DL_MG

Generalized Poisson equation
\[ \nabla \cdot (\varepsilon(r) \nabla \phi(r)) = -4\pi n(r) \]

Model system
(known solution)

64 MPI processes
2 OpenMP per MPI

Total execution time demonstrates \( O(N_{\text{grid}}) \) scaling up to \( \sim 1000^3 \) grid points
Aside: DL_MG

Source code features

- Written in modern Fortran and built using GNU Make
- Code organized into Fortran modules
- Combines distributed (MPI) and shared memory parallelism (OpenMP)

Modularity in practice

- Could have been part of ONETEP, but instead built an open-source library
- DL_MG can be utilized by other codes
- Has recently been interfaced with CASTEP [5, 8] & PSI4 [21, 22]

Source code available at www.dlmg.org
How do we manage day-to-day development?

Follow the key principles:

1. Version control
2. Testing
3. Documentation
4. Modularity
Version control

We have a Git repository hosted on Bitbucket and use a forking workflow:

- Single “official” server-side repository
- Private fork for each contributor
- Changes contributed via pull request
- Pull requests undergo review

New developers don’t need to worry about breaking the official repository!
We have a growing suite of regression tests:

- When new functionality is added, require a new regression test (“QC” test)
- Each test runs a calculation to produce known results (benchmarks)
- QC test suite is run by a Python program called `testcode`
- A test fails if outputs are outside of allowed tolerance (wrt benchmarks)

The test suite allows new builds of ONETEP to be validated.
Testing

We use the Buildbot package to automate testing of the software:

- A **Buildbot** instance regularly pulls the official master branch and starts a build:
- Each build includes compilation, code quality checks and QC test suite
- If any step fails, core developers are notified by e-mail

<table>
<thead>
<tr>
<th>Time</th>
<th>Revision</th>
<th>Result</th>
<th>Builder</th>
<th>Build #</th>
<th>Reason</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oct 27 07:10</td>
<td>be036a87e588...</td>
<td>success</td>
<td>cmth_gnu</td>
<td>#1068</td>
<td>The Nightly scheduler named 'nightly' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 27 03:32</td>
<td>be036a87e588...</td>
<td>success</td>
<td>cmth_intel</td>
<td>#1098</td>
<td>The Nightly scheduler named 'nightly' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 26 23:30</td>
<td>be036a87e588...</td>
<td>success</td>
<td>cmth_fort_omp</td>
<td>#1103</td>
<td>The Nightly scheduler named 'nightly' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 26 03:32</td>
<td>be036a87e588...</td>
<td>success</td>
<td>cmth_gnu</td>
<td>#1067</td>
<td>The Nightly scheduler named 'nightly' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 25 23:30</td>
<td>be036a87e588...</td>
<td>success</td>
<td>cmth_intel</td>
<td>#1097</td>
<td>The Nightly scheduler named 'nightly' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 25 23:30</td>
<td>be036a87e588...</td>
<td>success</td>
<td>cmth_fort_omp</td>
<td>#1102</td>
<td>The Nightly scheduler named 'nightly' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 25 03:32</td>
<td>be036a87e588...</td>
<td>success</td>
<td>cmth_gnu</td>
<td>#1066</td>
<td>The Nightly scheduler named 'nightly' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 24 23:30</td>
<td>be036a87e588...</td>
<td>success</td>
<td>cmth_intel</td>
<td>#1096</td>
<td>The Nightly scheduler named 'nightly' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 24 23:30</td>
<td>be036a87e588...</td>
<td>success</td>
<td>cmth_fort_omp</td>
<td>#1101</td>
<td>The Nightly scheduler named 'nightly' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 24 03:32</td>
<td>be036a87e588...</td>
<td>success</td>
<td>cmth_gnu</td>
<td>#1065</td>
<td>The Nightly scheduler named 'nightly' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 23 23:30</td>
<td>be036a87e588...</td>
<td>success</td>
<td>cmth_intel</td>
<td>#1095</td>
<td>The Nightly scheduler named 'nightly' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 23 23:30</td>
<td>be036a87e588...</td>
<td>success</td>
<td>cmth_fort_omp</td>
<td>#1100</td>
<td>The Nightly scheduler named 'nightly' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 23 03:32</td>
<td>be036a87e588...</td>
<td>success</td>
<td>cmth_gnu</td>
<td>#1064</td>
<td>The Nightly scheduler named 'nightly' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 23 00:48</td>
<td>be036a87e588...</td>
<td>success</td>
<td>cmth_intel</td>
<td>#1094</td>
<td>The Nightly scheduler named 'nightly' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 22 23:30</td>
<td>be036a87e588...</td>
<td>success</td>
<td>cmth_fort_omp</td>
<td>#1099</td>
<td>The Nightly scheduler named 'nightly' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 21 06:30</td>
<td>be036a87e588...</td>
<td>failure</td>
<td>cmth_fort_omp_dbg</td>
<td>#115</td>
<td>The Nightly scheduler named 'weekly' triggered this build</td>
<td>Failed testcode</td>
</tr>
<tr>
<td>Oct 22 18:50</td>
<td>be036a87e588...</td>
<td>success</td>
<td>cmth_gnu</td>
<td>#1101</td>
<td>The Nightly scheduler named 'nightly' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 22 15:11</td>
<td>be036a87e588...</td>
<td>success</td>
<td>cmth_intel</td>
<td>#1103</td>
<td>The Nightly scheduler named 'nightly' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 22 11:08</td>
<td>be036a87e588...</td>
<td>success</td>
<td>cmth_fort_omp</td>
<td>#1108</td>
<td>The Nightly scheduler named 'nightly' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 22 02:01</td>
<td>be036a87e588...</td>
<td>success</td>
<td>cmth_gnu_dbg</td>
<td>#170</td>
<td>The Nightly scheduler named 'weekly' triggered this build</td>
<td>Build successful</td>
</tr>
</tbody>
</table>
Documentation

Developer documentation:

- Template module file to encourage consistent documentation
- Each new module/procedure should be documented
- Encourage developers to comment their code as they write it

User documentation:

- User documentation is required for substantial new functionality
- Distributed with the code and at [www.onetep.org](http://www.onetep.org)
- Daussault Systèmes BIOVIA provide documentation for [Materials Studio](http://www.materialsstudio.com)
Modularity

ONETEP’s source code is organized as a loose hierarchy of modules:

▶ Modules encapsulate related functionality
▶ High-level modules depend on low-level modules
▶ Low-level modules provide basic functionality
▶ High-level modules perform sophisticated tasks

A module can often be developed without needing to consider the behaviour of the entire program.
Closing remarks
Summary

Building ONETEP

1. ONETEP is a tool for large-scale quantum simulation of matter

2. Development of ONETEP follows a few key principles

3. These principles keep ONETEP usable and maintainable
A thought: Research vs. software development

There is a tension between research outputs and good software development practice:

▶ Often we don’t know what the solution will look like until we create it.
▶ This “exploratory” development can make specification and testing difficult.
▶ Being overly strict may inhibit the creativity needed to solve scientific problems.
▶ **BUT** being too relaxed leads to unmaintainable nightmare code!

Scientific software development needs to balance developer freedom and creativity against software development rigour.
My thoughts on good scientific software development are based on my personal experiences.

Other people working in this field may disagree with what I have just said, or have a different emphasis.

Seek out a diversity of views and opinions on this topic!
Introducing ONETEP: Linear-scaling density functional simulations on parallel computers

DL_MG: A parallel multigrid Poisson and Poisson-Boltzmann solver for electronic structure calculations in vacuum and solution
Acknowledgements

Funding and compute resources

- EPSRC-funded CCP9 flagship (EP/P02209X/1)
- UK MMM Hub “Thomas” (EP/P020194/1)
- IRIDIS HPC Facility (Southampton)
- ARCHER UK National Supercomputing Service (UKCP consortium EP/P022561/1)

Thank you for your attention!

...any questions?


Bibliography III


