Software Engineering Support

Christopher Greenough, Alan Kyffin, Gemma Poulter

Software Engineering Group
Scientific Computing Department
STFC Rutherford Appleton Laboratory

christopher.greenough@stfc.ac.uk
http://www.software-support.ac.uk

Science & Technology Facilities Council

Opening comments…

- Scientists typically develop their own software because it often requires substantial domain-specific knowledge.
- As a result they typically spend 30% or more of their time developing software.
- However 90% or more of them are *self taught* and do not have a grounding in the basics of software development practices.

…consequences

- Much of existing scientific software is of low "quality" from the point of view of standards of "best practice" in professional software engineering (a bit harsh!).

- Those involved in developing codes feel justified in believing that they are of good "quality" because they generate valid scientific results that pass peer review.

- Can these statements be reconciled?

A definition…

Software Engineering is:

The use of processes and associated tools to write good quality software that produces excellent scientific results.

This is not a hard and fast, text book definition, it's just one we can work with!

Software quality…

- Users are generally happy with the results and publish papers in refereed journals, boosting confidence that the software is adequate.
- However, the ability to measure, report and improve software quality will have an impact on the users' perception.
- Being able to demonstrate, for example, that source code has been checked in a particular way, that a certain sort of testing has reached a quantifiable level, or that the introduction of new features has not caused any damage, will increase the users' confidence even further.

Collaborative computational projects…

- The Collaborative Computational Projects (CCPs) bring together leading UK expertise in key fields of computational research to tackle large-scale scientific software development, maintenance and distribution.
- Each project represents many years of intellectual and financial investment. The aim is to capitalise on this investment by encouraging widespread and long term use of the software, and by fostering new initiatives such as High End Computing consortia.
- They provide a software infrastructure on which important individual research projects can be built.
- They support both the R&D and exploitation phases of computational research projects.
- They ensure the development of software which makes optimum use of the whole range of hardware available to the scientific community.

Motivation…

- Much modern research requires software development
- This research software often has a very long life time
- EPSRC is keen that software developed under its funding is good quality and sustainable
- EPSRC has a software agenda
- The CCP program depends on good quality software
- We all would like to produce good quality software

CCPs…

CCP4	Prof David Brown	Macromolecular Crystallography
CCP5	Prof Stephen Parker	The Computer Simulation of Condensed Phases
CCP9	Prof Mike Payne	Computational Electronic Structure of Condensed Matter
CCP12	Prof Stewart Cant	High Performance Computing in Engineering
CCP-ASEArch	Prof Mike Giles	Algorithms and Software for Emerging Architectures
CCP-BioSim	Prof Adrian Mulholland	Biomolecular Simulation at the Life Sciences Interface
CCP-EM	Dr Martyn Winn	Electron Cryo-Microscopy
CCPi	Prof Phillip Withers	Tomographic Imaging
CCPN	Prof Geerten Vuister	NMR
CCP-NC	Dr Jonathan Yates	NMR Crystallography
CCPP	Dr Tony Arber	Computational Plasma Physics
CCPQ *	Prof Tania Monteiro	Quantum Dynamics in Atomic, Molecular and Optical Physics
CCP-SAS	Prof Steve Perkins	Analysis of Structural Data in Chemical Biology and Soft Condensed Matter
CCPForge	Prof Chris Greenough	Collaborative Software Development Environment Tool

Some scenarios…

- A legacy code that is still require by the community and needs further developments
- The need to use an existing routine in a new development.
- Engaging in a multi-site collaborative development.
- The disc crash – my laptop had the only copy!
- The need to improve the testing of my code – but it take such a long time.
- How do I demonstrate the quality of my code?

Common situations…

- Many practitioners of scientific software development will acknowledge (though perhaps not on the record) that the vast majority of their code started life in a research project where the results were all that mattered.
- These codes have never been subject to formal analysis or testing.
- The code was passed from research student to post doc and back again each time developing the functionality to meet a specific scientific goal.
- It move from machine to machine and things added and removed as compilers and system features change
- There has not been nor will be any documentation apart from the few comments in the code.

Common results…

With code built on such foundations and developed in a piecemeal fashion, it is likely that there are problems in several areas, including:

- Undiscovered bugs.
- Duplication of numerical algorithms with different implementations.
- Unused code and variables.
- No overall documentation.
- No consistent style to aid new programmers.
- Code written in order to tweak one data structure into another or misusing data structures.
- No overview of intended route through the code.
- No documentation of (in)compatible input data.

Things we would like to improve…

- Establish good software development practices
- Capture/rescue important legacy software
- Improve the quality of our software base:
  - language conformance
  - portability
  - design and structure
  - testing
  - maintenance
  - improve software quality
- Where possible use software engineering tools to automate and make the process easier and more measurable
- Ensure our developer adopt some good practice and develop their own process of improvement!

Software Engineering Group…

Encouraging the development of high quality and sustainable software by the use of software engineering tools and techniques

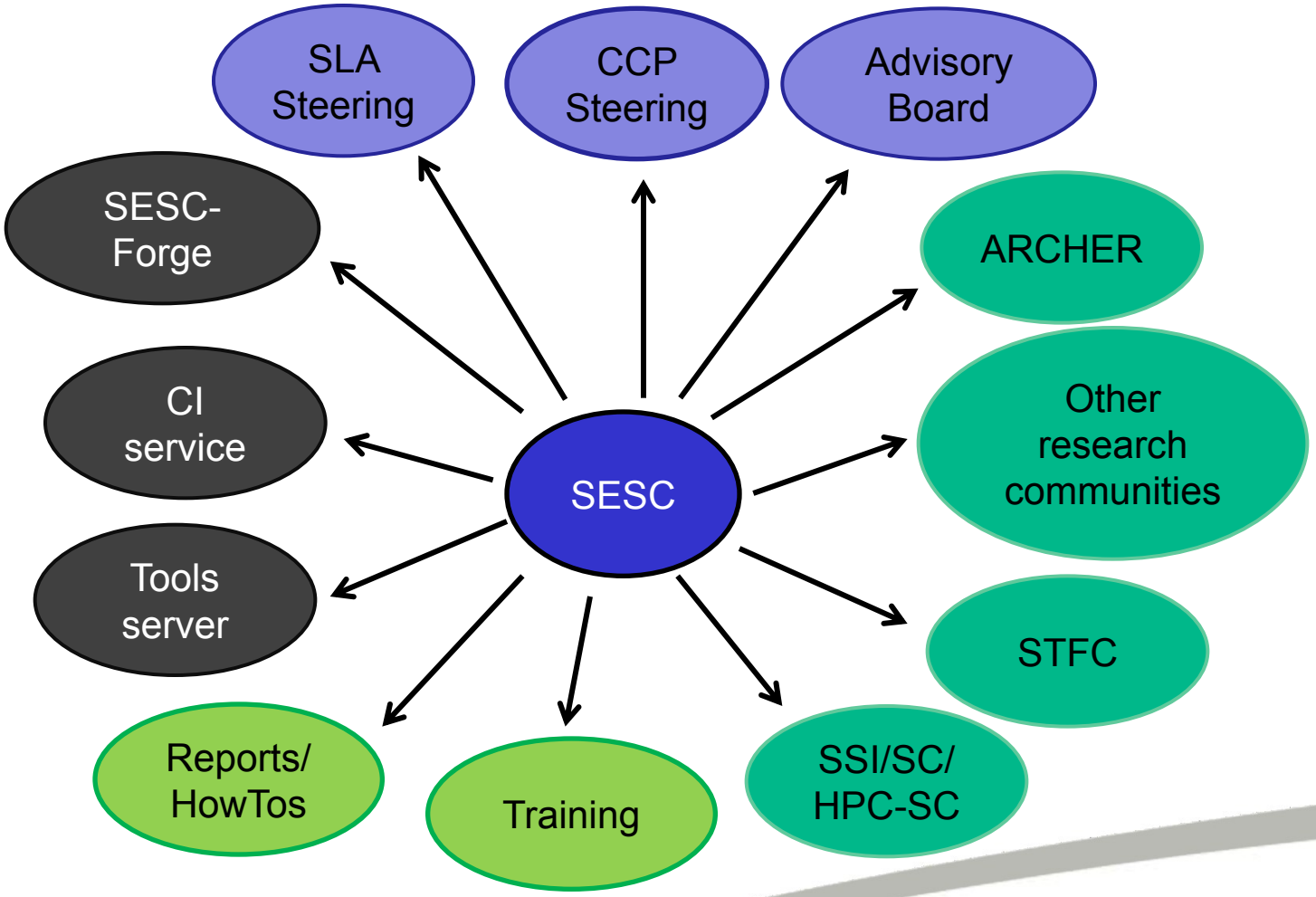
The Software Engineering Group's main activities centre around practical and applicable software engineering. We aim to make our suggestions pragmatic and not dogmatic.

- Practical software engineering for computational science
- Software Quality Assurance (QA): methods and tools
- Software re-engineering/re-factoring and transformation
- Automated testing and continuous integration
- The Software Engineering Support Centre (SESC)

SESC…

- History
  - Started by JISC as CCPForge
  - Funded by EPSRC under SLA as SESP
- New 5 years of funding by EPSRC from August 2013
- Continue to promote good software engineering in UK scientific computing community:
  - Pragmatic
  - Processes
  - Tools
- Through – provision, education and practice

SESC Map…


SLA Steering
CCP Steering
Advisory Board
SESC-Forge
ARCHER
CI service
Other research communities
SESC
Tools server
STFC
Reports/ HowTos
Training
SSI/SC/ HPC-SC

Our goal…

- Enable computational scientists to recognise the usefulness for software engineering in their work.
- Introduce ideas from software engineering.
- Describe and provide some of these tools that can be useful in adopting these ideas.
- Enable development teams to produce their own working practices to improve software quality and increase productivity.

SESC Activities 1…

- Maintain and develop CCPForge
  - Access revision control system
  - Access to QA tools
  - Continuous integration/testing
- Continuous Integration (CI)
  - Make running tests easy
  - Quickly learn when something is wrong
  - Access to variety of hardware, OS, compilers
- QA tools server
  - More advanced usage than CCPForge

SESC Activities 2…

- Training
  - Introduction to software engineering process
  - Use of revision control
  - Tools workshops, including hands-on sessions
- Seminars
  - SESC, software engineering, CCPForge
- Advice
  - Processes, re-engineering, documentation
- SE advice and input to grant applications

Software quality…

- Quality criteria include but are not limited to:

Economy	Correctness	Resilience	Integrity
Reliability	Usability	Documentation	Modifiability
Clarity	Understandability	Validity	Maintainability
Flexibility	Generality	Portability	Interoperability
Testability	Efficiency	Modularity	Resuability

- Because there are so many criteria, we can not use them all for measuring software quality.
- Some of the desired characteristics can be used in software product standards as guiding actual development work.
- To be a high quality software, software must be able to run correctly and consistently, have few defects (if there are), handle abnormal situation nicely, and need little installation effort.

Software Metrics…

- Today there are many ways of measuring the complexity of a program, the measures included in this tool are typical of those which have been developed over the past few years.
- Software complexity is a way of identifying, classifying and measuring various features and characteristics of a piece of code which may when considered lead to changes in code which will ultimately lead to better code and lower the lifetime cost of the product.
- The first major paper published on the subject was in 1976 by McCabe[1] with his description of a measure for the cyclometric complexity of a program, a very simple measure to calculate.
- Shortly afterwards in 1977 a book published by Halstead [2] described ways of calculating new measures which analysed the structure of software in more detail and allowed the calculation of development time and costs.

Some observations…

- There is no one magic process, method or tool
- Such transformations require the developer to be involved
- The developers must be engaged for improvement to take place
- A process for migrating legacy Fortran software has been defined and some software tools identified – it does work with help
- Compilers can be good tools in improving and maintaining software
- A software tools resource has been started – these cost money!
- A web interface to some of the tools has been written
- There are software tools to aid migration of codes
- There are tools to help in the understanding and documentation of software
- All these tools require a short learning curve
- Providing tools for testing and software quality is a challenge!

About you and your software…

- Do you have a formal software development and improvement plan?
- Are you getting fewer bug reports as the software develops?
- Is adding new functionality easier?
- What is the learn curve like for a new developer?
- Can you port to a new system more easily?

Challenges and Difficulties…

- Provision of Software Tools
  - there are a limited number accessible tools
  - many tools are particularly intolerant for mixed code
  - tools generally do not recognise pre-processors
  - language conformance is generally covered – a few problems
  - transformations are limited
  - The available tools often create too much output which is difficult to navigate and interpret
- People and Processes
  - Changing any development process is long term – years not months
  - There are very few short term gains to be seen
  - Using addition methods and process takes addition time and effort
  - Author recognition is a serious problem
  - The benefits of using these additional processes are often not immediately obvious

SESC - our process…

- Make use of what is being done already
- Focus on useful ideas not a process
- Provide tools: licenced, bespoke, public domain
- Provide advice
- Be involved in development projects
- Make use of these methods

Some links…

- Scientific Computing Department

  www.stfc.ac.uk/scd
- Software Engineering Support Centre

  www.softeng-support.ac.uk
- CCPForge

  ccpforge.cse.rl.ac.uk
- Software Sustainability Institute

  www.software.ac.uk
- Software Carpentry

  software-carpentry.org

Contact details…

Prof Chris Greenough
Software Engineering Group
STFC Rutherford Appleton Laboratory
Harwell Oxford
Didcot
Oxfordshire OX11 0QX

Tel: 01235 445307
Email: christopher.greenough@stfc.ac.uk

SESC: http://www.software-support.ac.uk