EMMC-Webinar
on
Best Practices for Software Development

Volker Eyert, Materials Design and Kurt Stokbro, Synopsys

29 May 2018

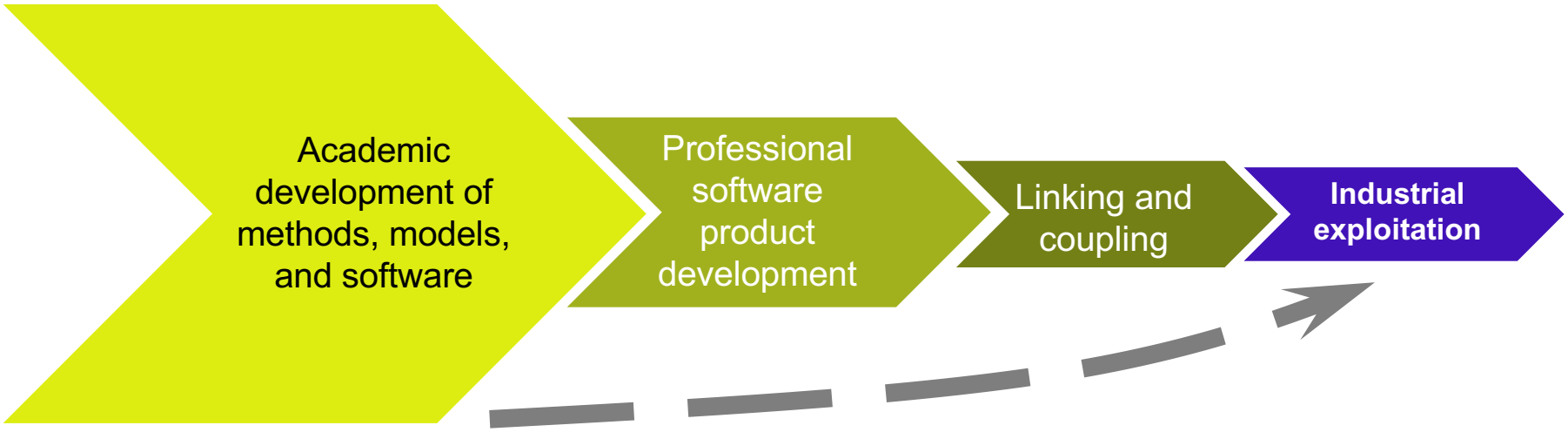
This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 723867

The European Materials Modelling Council

Agenda

- Introduction
- Scope
- Model description and software architecture
- Programming language and deployment
- Intellectual property and license considerations
- Testing, verification, validadation and robustness
- Organization of software development
- Version control
- Metadata
- Documentation
- Support
- Conclusion

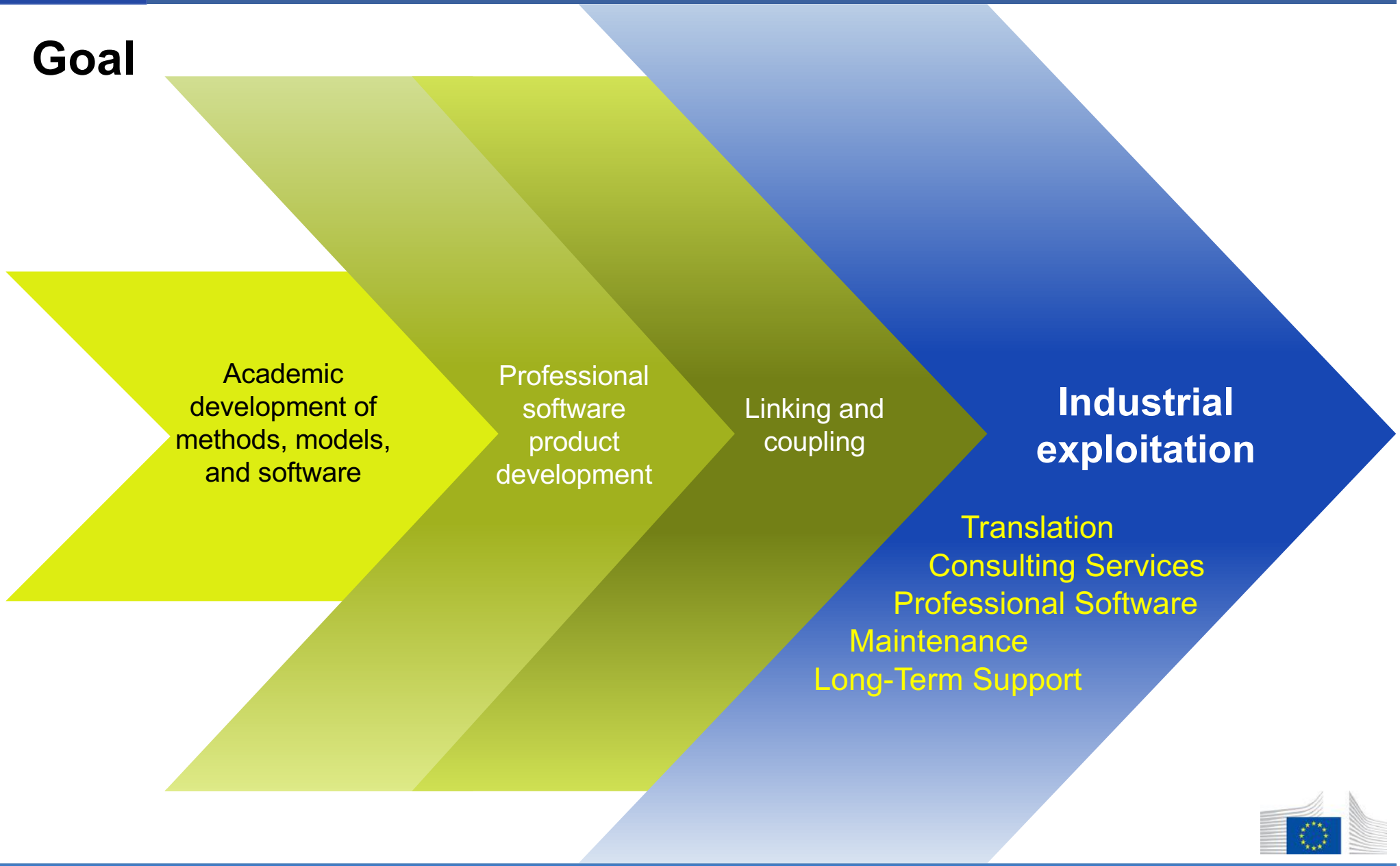
Current Deployment of Academic Software



The European Materials Modelling Council

Industrial Software Deployment

Goal



The European Materials Modelling Council

Introduction

- EMMC-CSA (Coordination and Support Action):
  - Installed in 2016 by European Commission
  - Foster materials research in European industry
  - Stimulate materials modelling as integral part of industrial R&D
  - Combines expertise of academic and industrial partners
  - Strengthen link between (academic) software developers, distributors, and (industrial) end users
- White paper/webinar on software development:
  - Provide academic researchers with guidelines for sustainable modelling-software development
  - Increase awareness of importance of early design decisions
  - Share best practices of software development

The European Materials Modelling Council

Scope

- Huge amounts of scientific software developed over decades:
  - Mainly in academia, mainly by students and postdocs (who soon after left), short time horizon
  - Often “Quick and dirty” coding, lack of documentation
  - Less emphasis on IP and licensing considerations
- Increasing degree of commercialization:
  - Quantum chemistry codes → chemical and pharmaceutical industry
  - Companies like Biovia (Biosym, MSI, Accelrys), Materials Design, Schrödinger, QuantumWise/Synopsys, SCM establishing value-added chain from academic research to industrial deployment
- Software-development guidelines for academic developers
  - More rapid integration of academic software with industrial software

Model Description and Software Architecture

- Early planning:
  - Scope of the software
  - Vision for future developments
- Software Architecture:
  - Modularization, well-defined interfaces
  - Use libraries (*e.g.* LAPACK) whenever possible
  - Try to combine selected software modules into a library that can be shared and maintained by several groups (*e.g.* libxc)

The European Materials Modelling Council

Programming Language and Deployment

- Fortran (I-IV, 66, 77, 90, 95, 2003, 2008, 2018):
  - Probably most used language in scientific computing
  - Tremendous move from “spaghetti code” to modern object-orientation
  - Upward-compatibility saves investment in decades of code development
  - Good performance, many compilers available (free and commercial)
- C++:
  - Object oriented, requires some computer science skills to master
  - Increasingly used for scientific applications
  - Good performance, compilers available
- Python:
  - Increasingly popular in scientific community
  - Interpreter, fast prototyping, less performant, errors to be found at run time
  - Can be interfaced with compiled language at increased complexity

The European Materials Modelling Council

Programming Language and Deployment

- Parallel programming:
  - MPI
  - OpenMP
- GPU programming:
  - CUDA
  - OpenACC
- Coding guidelines
  - Increase readability and improve work in a development team
  - Try to use community standards and be consistent
- Source-code documentation
  - At the disposal of single developer but strongly (!) recommended
  - Can/should be done according to rules defined by development team
  - Allows for automatic documentation through Doxygen or Sphinx

Intellectual Property and License Considerations

- Legal matters:
  - not exciting to deal with but utterly important
  - decisions at an early stage of crucial importance, especially when many people are involved
- Ownership:
  - employer or employee?
- Intellectual property rights:
  - copyright, patent?
  - national laws, international regulations?
- License models:
  - permissive, copyleft, proprietary?

Intellectual Property and License Considerations

Permissive licenses, free and open-source software (FOSS):

- Free software (Free Software Foundation, 1986):
  - Freedom 0: The freedom to **run** the program for any purpose
  - Freedom 1: The freedom to **study** how the program works, and change it to make it do what you wish
  - Freedom 2: The freedom to **redistribute** and make copies so you can help your neighbour.
  - Freedom 3: The freedom to **improve** the program, and release your improvements (and modified versions in general) to the public, so that the whole community benefits
- Open Source Definition (Open Source Initiative, 1998):
  - Based on Debian Free Software Guidelines
  - Largely equivalent
- others: Berkeley Software Distribution, MIT, Apache, Educational Community License

Intellectual Property and License Considerations

Copyleft licenses:

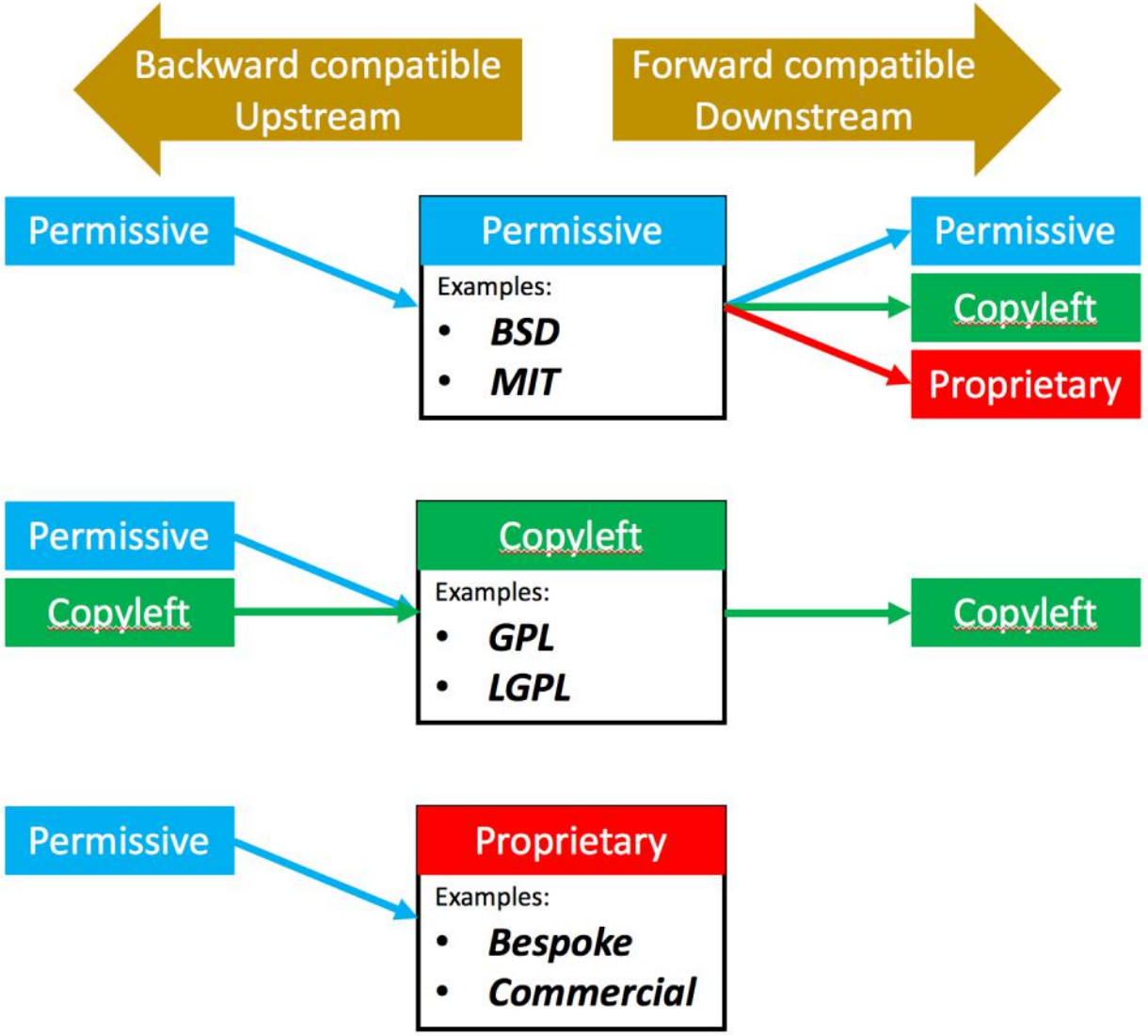
- apply to free and open source software
- modified versions must preserve freedom 0-3 (viral licenses)
- examples
  - GNU General Public License (GPL)
  - GNU Lesser General Public License (LGPL)
  - Mozilla Public License (MPL)

Proprietary licenses:

- include constraints (no source code, license fee)
- often come with commercially developed software
- often come together with long-term support

Intellectual Property and License Considerations

License schemes
- Combination
- Directionality



A. Morin *et al.*, PLoS Comp. Biol. **8**, e1002598 (2012)

Intellectual Property and License Considerations

Copyright and licensing in practice: first steps

- Add Copyright line to each file
  - Version MyCode 29.05.2018 MyFirstName MyLastName
  - Copyright © 2018 MyFirstName MyLastName
  - Please see file MyCopyrightFile for details.
- Create a MyCopyrightFile
  - Include GPL/LGPL as available from gnu.org
  - Alternatively include BSD, MIT, … license files available from the respective sources

Testing, Verification, Validation, and Robustness

- Testing:
  - Includes all parts of a software
  - Considers different platforms (hardware architectures, operating systems)
  - Ensures consistency of results obtained from different releases
  - Ideally done at a hierarchy of time intervals and complexities
- Verification:
  - Compare with other implementations or with prototypes of less complexity
- Validation:
  - Comparison of calculated results to experimental data
- Robustness:
  - Make sure that the implementation does not only work for simple test systems

The European Materials Modelling Council

Organization of Software Development

- Scrum model:
  - Iterative development by a team of developers in short development cycles (sprints), typically 3-week cycles

Phases:

- Planning
- Sprint
- Demo
- Retrospective

Version Control

- Version control systems:
  - Record all changes to the code
  - Archive and allow recovery of all previous states of the code
  - Coordinate parallel editing of the same piece of code by several developers
  - Allow simultaneous work on different branches of the code
  - Allow merging different branches of the code
  - Centralized vs. distributed systems
- Basic functionality of VCS:
  - Check out from central repository to local working directory of developer (copy/lock vs. copy/merge systems)
  - Local work of developer
  - Check in to central repository (with time stamp and user ID)
  - Synchronization with other developers work

The European Materials Modelling Council

Metadata

- Much underrated and rarely considered
- Crucially important and prerequisite for reproducing results
- Should include
  - Version/release number/release date
  - Copyright and licensing information
  - development group/developer
  - Compiler and compiler options
  - Hardware used to run a calculation
  - User running the calculations and working directory
  - Start and end time of calculations
  - Complete list of all input data
  - all information in all intermediate/output files (check internal consistency)

The European Materials Modelling Council

Documentation

- Model description
- Design guidelines (code architecture, accuracy, limitations)
- Technical documentation of implementation using, e.g., sphinx or doxygen
- User documentation (user's guide)
- Video tutorials
- Marketing (raise initial interest, address decision makers)

User Documentation

- Installation guide: first contact, clear, mention auxiliary software
- Case studies and workable examples
- Complete list of all variables (default values, physical units, criticality, sensitivity of results to changes)
- Complete list of all input/intermediate/output files (role, size)
- Graphical user interface (screenshots)
- Background information on
  - Model description and algorithms
  - Software architecture and implementation
  - Limitation
  - Selected input parameters

The European Materials Modelling Council

Support

Support at its best

- Long-term stability
- Long-term maintenance (updates, enhancements)
- Hotline, 24h response
- Connect to industrial software owners

Conclusion

- Early design decisions are mandatory:
  - Scope of software, vision of future development
  - Model description, software architecture, programming language
  - Organization of software development process
  - License model (proprietary, permissive, copyleft)
  - Version control, source-code documentation
  - Testing, verification, and validation
  - User documentation and support

The European Materials Modelling Council

www.emmc.info

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 723867