UMONS
Université de Mons

POLYTECH MONS

General guidelines and discussion of open source programming in critical applications

Illustration with R-testbench, a Python package for remote instrumentation

Ir. Alexandre QUENON (UMONS)

Thanks to fnrs FREEDOM TO RESEARCH inforTech numediart for their support.

Many situations and applications are partly or totally handled by programming short scripts or full software solutions.

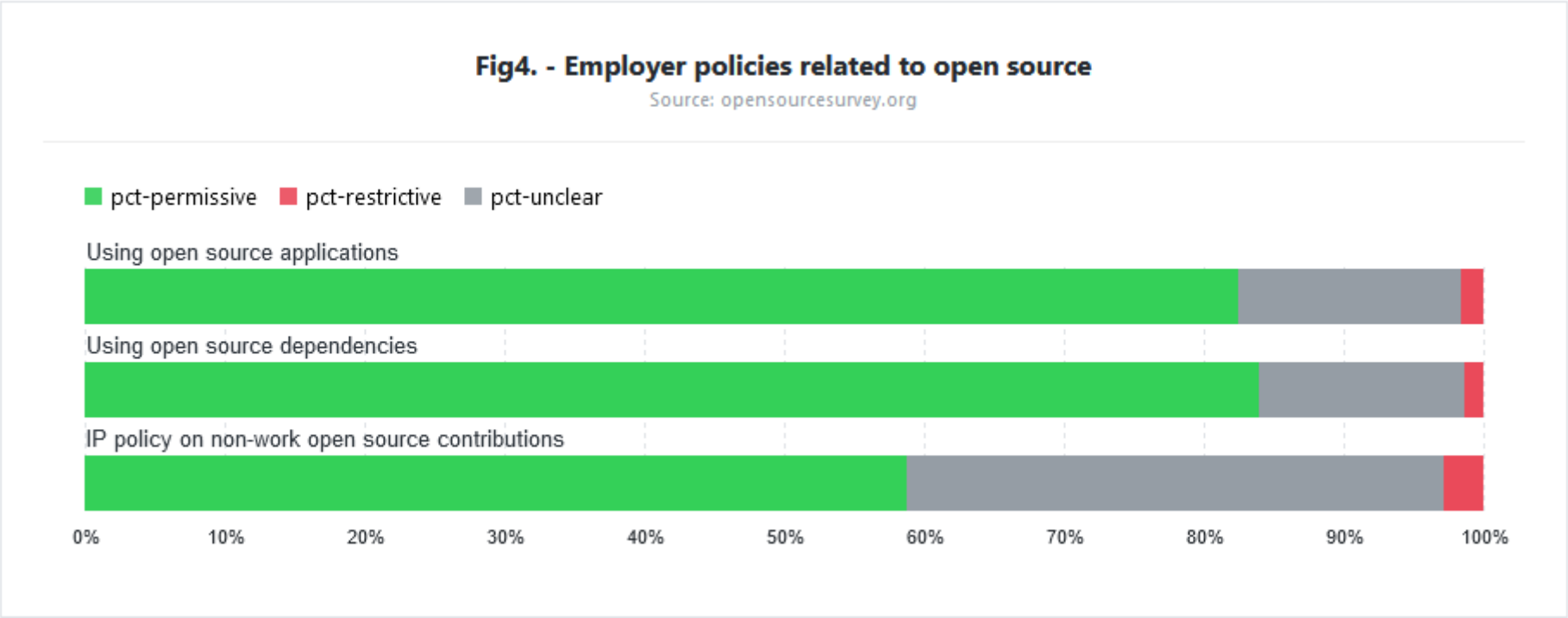
A non-exhaustive list of examples:

- instrumentation,
- data processing,
- simulations of complex phenomena.

Open source has become a major player at all levels of research and industrial software development.

- Exponential growth of the open source over the past decades.

- Increasing adoption by private companies (2017 survey).



The purpose of this presentation is to propose and discuss some guidelines for open-source programming.

Why?

- Even though the fast and continuous expansion, open source lacks clear guidelines for management, strategies and applications.

- Such guidelines can only be achieved by experience, which must be shared between users and developers.

How?

- Four statements will be provided as general rules.

- They will be illustrated with the design of R-testbench, a software solution that performs remote.

Rule #1: identify the needs.

Rule #2: take time for benchmarking.

Rule #3: select the paradigms and the appropriate language.

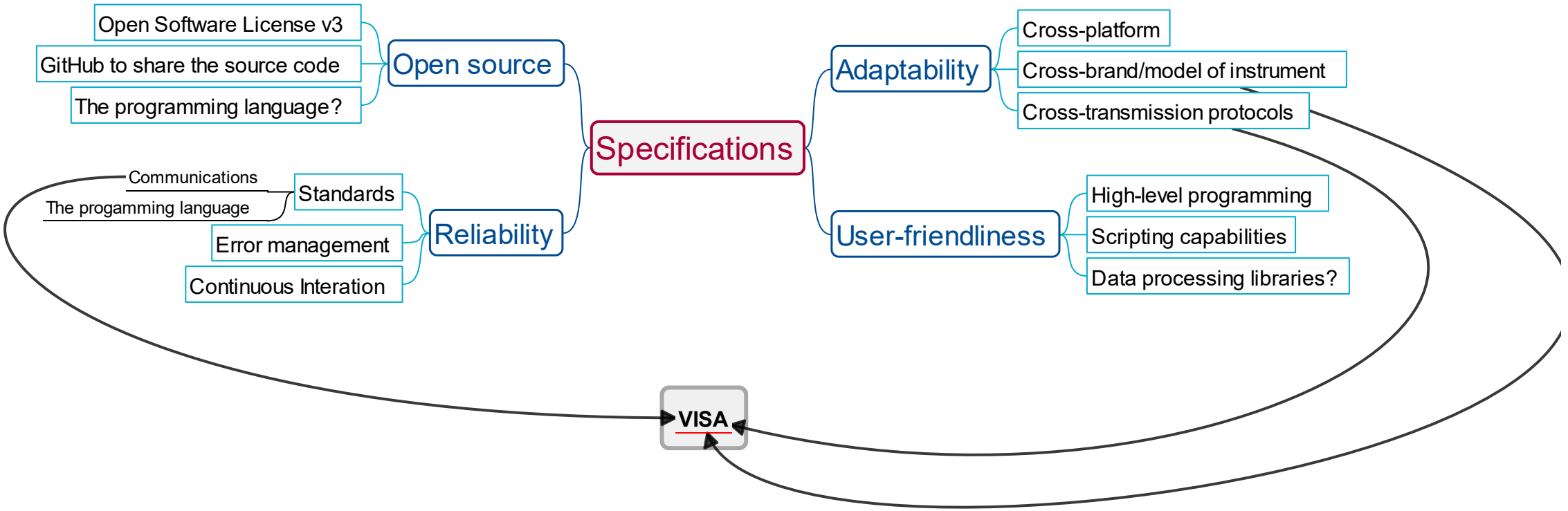
Rule #4: design the software architecture.

Rule #1: identify the needs.

There are three effective ways to do it.

- Define specifications with specialists.
- Be aware of the habits in the target field of applications.
- Learn from real use cases.

Requirements for a modern software tool in instrumentation.



Rule #1: identify the needs.

Rule #2: take time for benchmarking.

Rule #3: select the paradigms and the appropriate language.

Rule #4: design the software architecture.

Rule #2: take time for benchmarking.

There are common traps for engineers and scientists that must be avoided.

- The need to design their own tool, because they can.
- The desire to be free from “any bias”, for example coming from inspiration from competitors.
- The certainty to do better, because trust them, they are engineers.

Main characteristics of software solutions for measurement instruments remote control and automation.

Tool	License	Programming			VISA compatible protocols	Automatic recognition	Supported result files types
		Level	Language	Graphical			
LabVIEW	proprietary	**high** or low	MathScript	✓	✓	X	**many**
MATLAB	proprietary	low	**Matlab**	✓	✓	X	**many**
Octave	GPL	low	Octave	X	X	X	**bin, HDF5, mat, csv, txt**
Scilab	GPL	low	Scilab	X	✓	X	SOD (HDF5)
PyMeasure	**MIT**	**high**	**Python**	X	✓	X	csv

Rule #1: identify the needs.

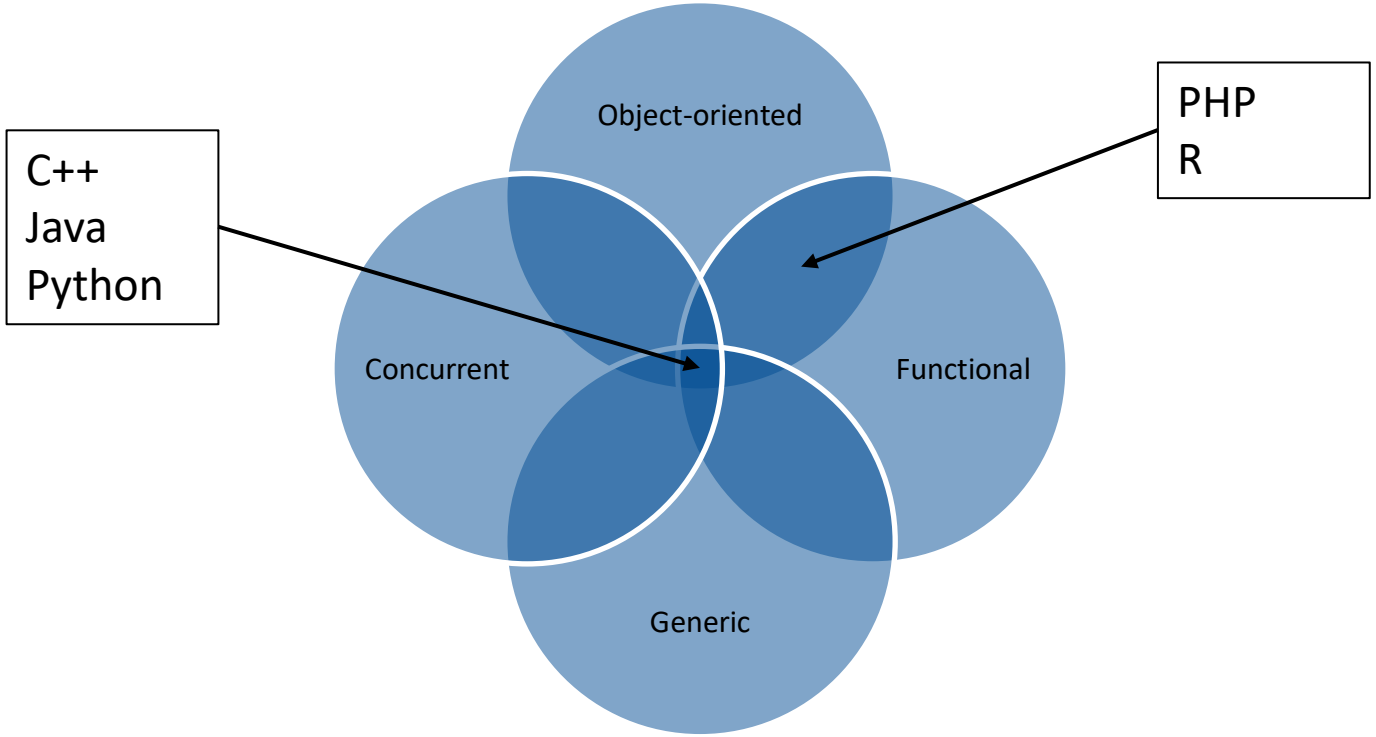
Rule #2: take time for benchmarking.

Rule #3: select the paradigms and the appropriate language.

Rule #4: design the software architecture.

Rule #3: select the paradigms and the appropriate language.

Based on the requirements and the characteristics identified with the benchmarking, the paradigms can be selected, and then the programming language.



Python has been selected has the main programming language for R-testbench.

Object-oriented (encapsulation)

- Front end calling the VISA libraries.
- Automatic instrument recognition.
- Rich IO interface to output results.

Scripting capabilities

- Creation of scripts for specific test benches.
- Data processing and numerical computations libraries.

Reliability

- Standardization.
- Error management system.
- Test framework.
- Strong community.

Rule #1: identify the needs.

Rule #2: take time for benchmarking.

Rule #3: select the paradigms and the appropriate language.

Rule #4: design the software architecture.

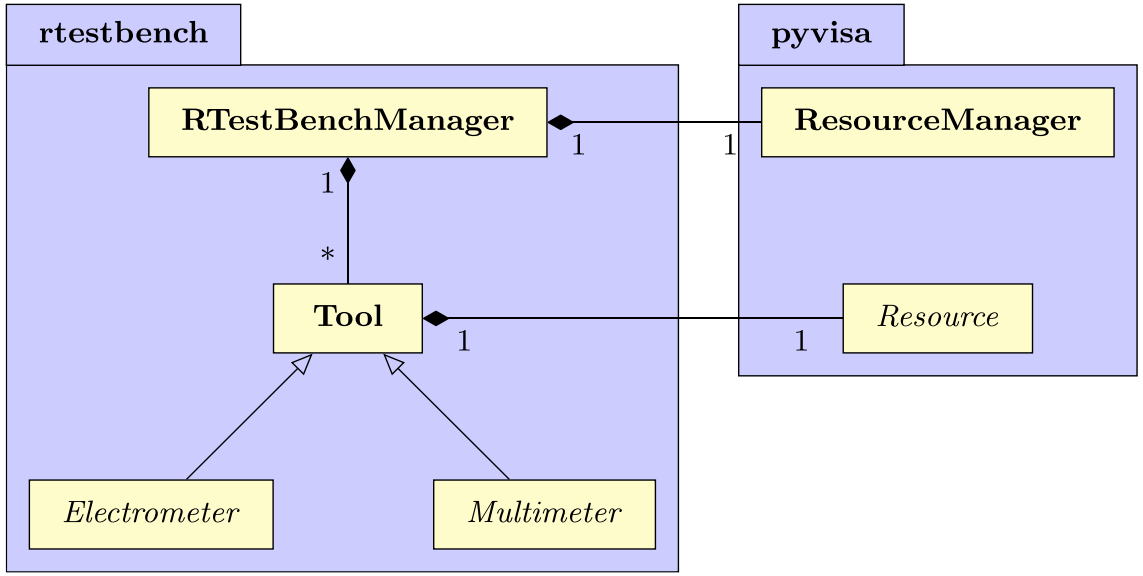
Rule #4: design the software architecture

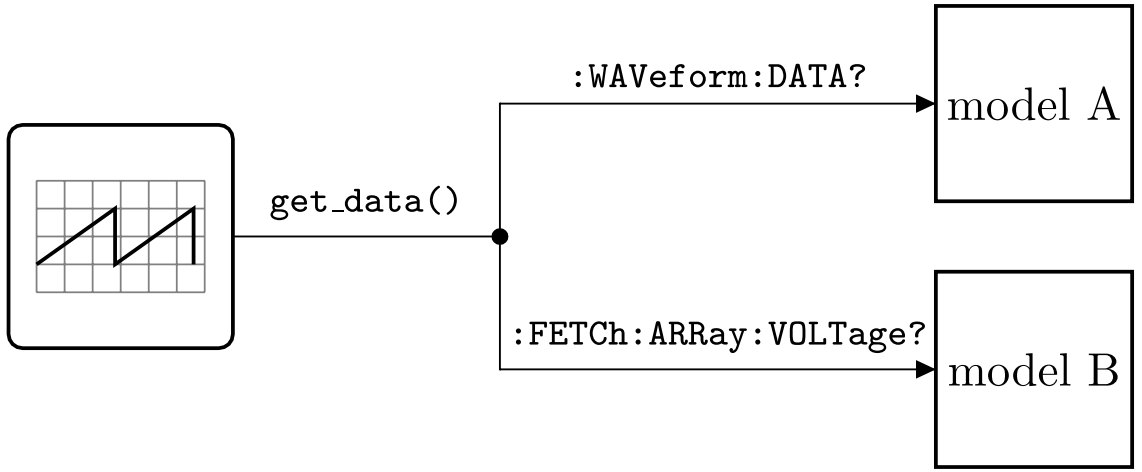
Some people say that this work is an 80/20% split between:

- 80% working on a paper or a board to create the software,
- 20% working on a computer to actually code.

My experience is that this might not be true regarding the timing, but with design.

R-testbench's architecture provides high-level generic and specific interfaces that encapsule SCPI*.





*SCPI = Standard Commands for Programmable Instruments.

Four “rules” to remember as general guidelines.

Rule #1: identify the needs.

Rule #2: take time for benchmarking.

Rule #3: select the paradigms and the appropriate language.

Rule #4: design the software architecture.

Open-source programming keeps on growing and offer challenging perspectives for developers and users.

The main perspectives concern:

- the management and methodologies;
- the applications, that can either have a lack of open-source tools or require to enhance or replace outdated software.

The proposed guidelines have been applied to design the R-testbench Python package for software instrumentation.

References

- Slide 3, figure – Open Source Survey (https://opensourcesurvey.org/2017/).

- Slide 10, table (adapted from) – Alexandre QUENON, Evelyne DAUBIE, Véronique MOEYAERT, and Fortunato DUALIBE, “R-testbench: a journey in open source programming for remote instrumentation with Python”, in *Sensors & Transducers*, submitted.

- Slide 16, figures – idem.

UMONS
Université de Mons

POLYTECH MONS

General guidelines and discussion of open source programming in critical applications

Illustration with R-testbench, a Python package for remote instrumentation

Ir. Alexandre QUENON (UMONS)

Thanks to fnrs FREEDOM TO RESEARCH inforTech numediart for their support.