THE ELT LINUX DEVELOPMENT ENVIRONMENT

F. Pellegrin, C. Rosenquist
European Southern Observatory

https://www.eso.org
The ELT

Extremely Large Telescope

- 39m ground-based
- Cerro Armazones
- First stone May 2017
- First light expected 2024
- Largest optical/near-IR
- Exoplanets, star formations, protoplanetary systems

- Five-mirror design
- M1: 798 segments 1.4 meters wide 5cm thick (3 PACT, 6 ES, 12 WH)
  - Figure loop at 500Hz ~ 1Gbit/s traffic
- M4: 4 meters (~6000 actuators)
- Alt-azimuth mount with 6 LGS
Software challenges

- Components of very different scope:
  - Real-time performance
  - High level data handling and post-processing

- Long time project (> 30 years)
  - Maintenance

- Different developer base:
  - In-house / external
  - Engineers / scientists
DevEnv Overview

C/C++
gcc (Clang)
cpplint
Cppcheck
Googletest
gcov

Python
Anaconda
pylint
unittest
nose
tests

Java
OpenJDK
Checkstyle
findbugs
TestNG / Mockito

Qt5

Doxygen

Eclipse

CentOS

waf

Gdb, strace, valgrind,
Systemtap, htop, tuned

Protobuffers
DDS
ZeroMQ
OPCUA
...

Build system challenges

- Single build system for C++ / Python / Java
  - Reliable partial builds
  - Full parallelization
  - Requires less specific knowledge

- Automatic dependency management

- Efficient and parallel

- Off-tree builds

- Ease of integration with new tools

- Logging and debugging support
Open source project started in 2005

Entirely Python based (2.5 -> 3.6)

Focus on:
- Portability
- Speed of execution

Efficiency on condition of rebuilds

Supports many languages and tools; expandable

Users: Samba, RTEMS, Ardour, game companies
waf

- wscript: build scripts defining configuration, options and build steps
  - Python code
  - Interaction with the waf framework

- Command line execution of phases
  - configure
  - build
  - test
  - install / dist
  - Custom commands
def options(opt):
    opt.load('compiler_cxx python pyqt5 ')

def configure(conf):
    conf.load('compiler_cxx python pyqt5 ')
    conf.check(header_name='stdio.h', features='cxx')
    conf.check_python_version((3,5,0))

def build(bld):
    bld.shlib(source='a.cpp inc/a.h', target='alib', export_includes='inc')
    bld.program(source='m.cpp', target='app', use='alib')
    bld.stlib(source='b.cpp', target='foo')
    bld(features="py pyqt5", source="src/test.py src/gui.ui",
         install_path="${PREFIX}/play/", install_from="src")
wscripts are readable and easy but still…

- wtools as a layer for:
  - Simplification for common tasks for users
  - Centralized maintenance and roll-out of new features
  - Easier to enforce certain practices

Can reduce script to a single line:

```python
from wtools.module import declare_cprogram
declare_cprogram(target="foo", use="bar")
```

- Tasks for primary artifacts and additional ones are created: tests, installation, linting …
wtools

- Based on set on conventions:
  - Directory structure, file positioning, file naming

- Currently supporting:
  - C/C++ program, shared and static library,
  - Python program and package,
  - Qt5 C++ or Python program
  - Java JAR packages.

- Custom modules that leverage full waf can be created for specific needs not included in wtools
Future challenges

- Early adoption with feedbacks
  - Implementation of new requests is easier
  - Very efficient resource-wise

- We need to help users to adapt to this new technology and maintain it actively to meet expectations

- What else we are looking at:
  - Containerization (Docker and LXC)
  - Deployment of applications