# A Framework for Quality Assessment of ROS Repositories

2016 IEEE/RSJ International Conference on Intelligent Robots and Systems

André Santos

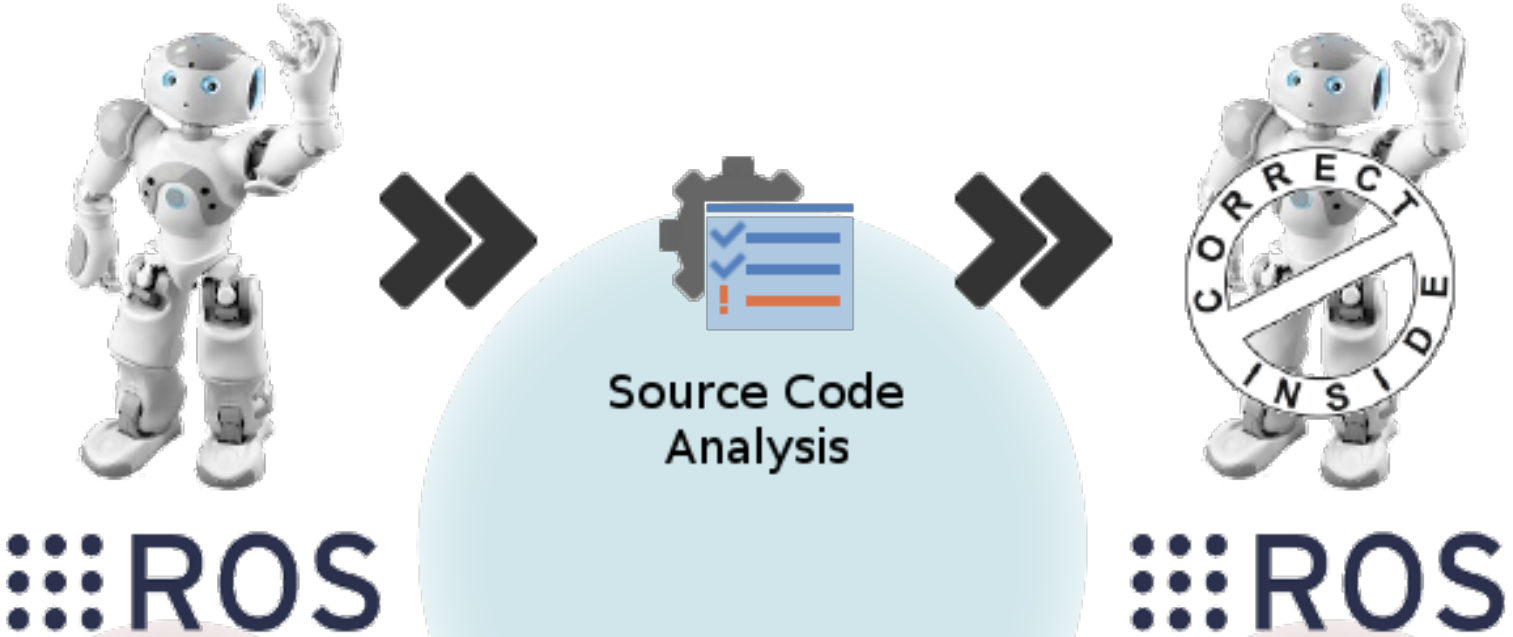
Alcino Cunha  Nuno Macedo  Cláudio Lourenço

INESC TEC & Universidade do Minho, Portugal

October 13th, 2016

# Software in Robotics

Challenge: many interesting robot applications (e.g. health, industry) require high levels of safety and flexibility. These come from software – high-quality software.




We analysed some popular robots, and produced a framework to automate the process.

# Software Quality

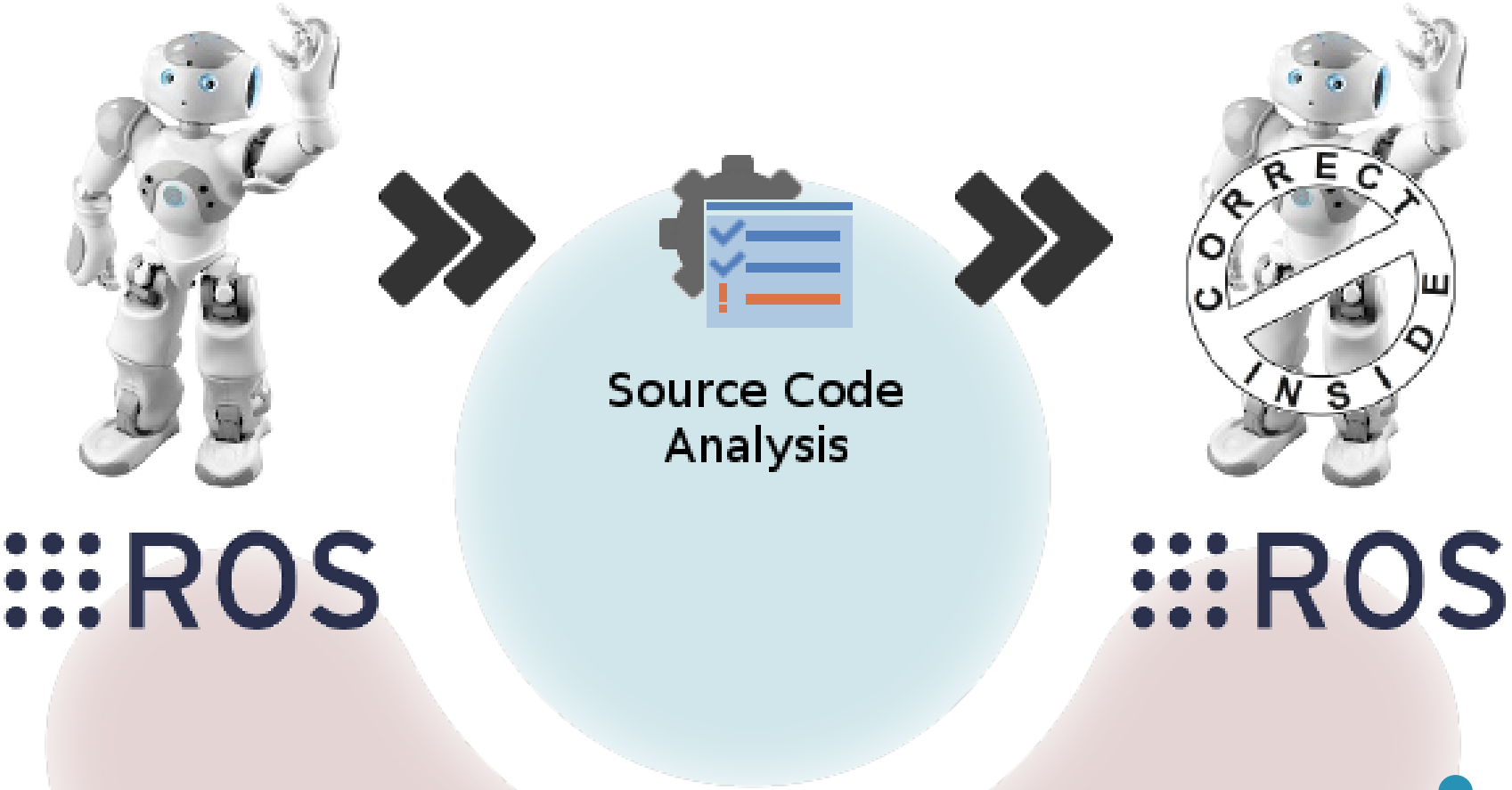
One way to minise safety issues is to produce high-quality software.

To improve safety and quality, adopting coding standards – rules and recommendations about how to write the software – is a common practice (e.g. ROS C++ Style Guide, MISRA C++, HIC++).

Another common technique is to analyse quality metrics – numeric values about how much a property manifests (e.g. lines of code, number of dependencies, function complexity).

# The HAROS Framework: Overview

The HAROS Framework (High-Assurance ROS) aims at providing an analysis platform for ROS systems, making robots more reliable.



# The HAROS Framework: Common Questions

Q: Why not use SonarQube, Eclipse or . . . instead?

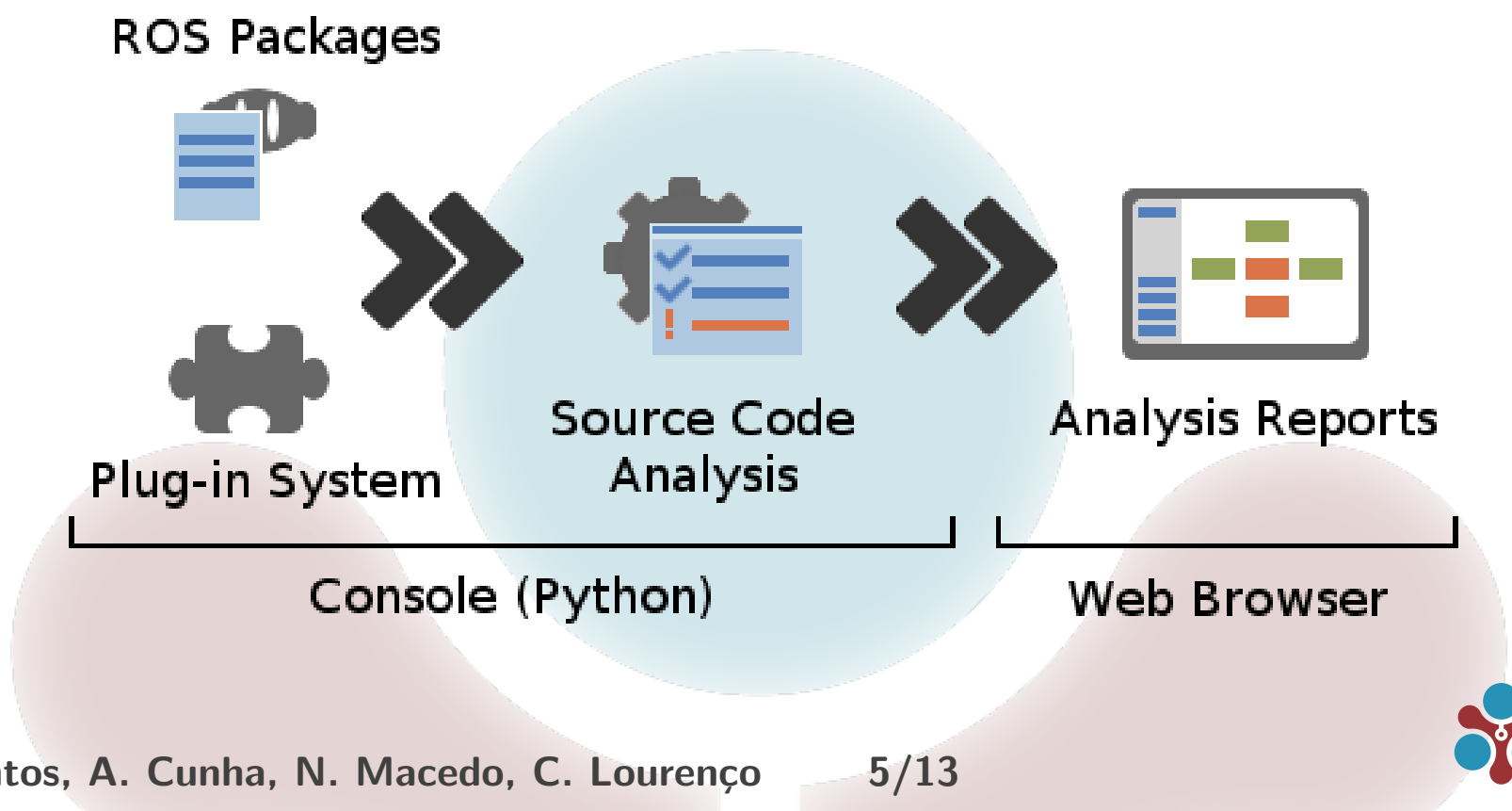
A: HAROS is free, extensible, ROS-oriented and platform-independent.

Q: Why does robotics software need its own tool?

A: There are some analyses that we can explore in more detail: configurations, software role, models, etc.

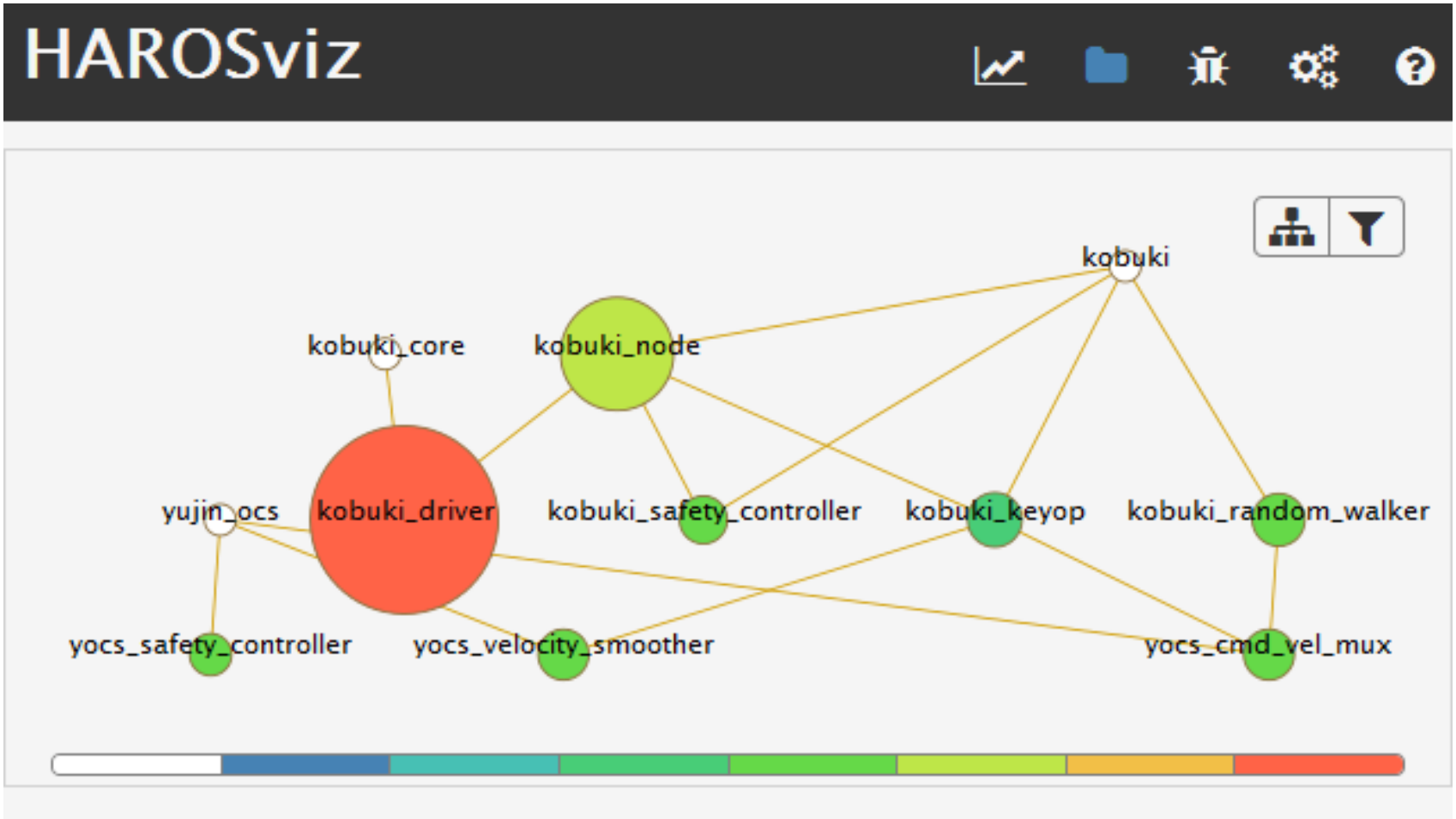
# The HAROS Framework: Main Features

- Source code fetching of indexed ROS packages.
- Plug-ins enable integration of third-party analysis tools.
- Interactive graphic reports of the results mirroring the ROS architecture.



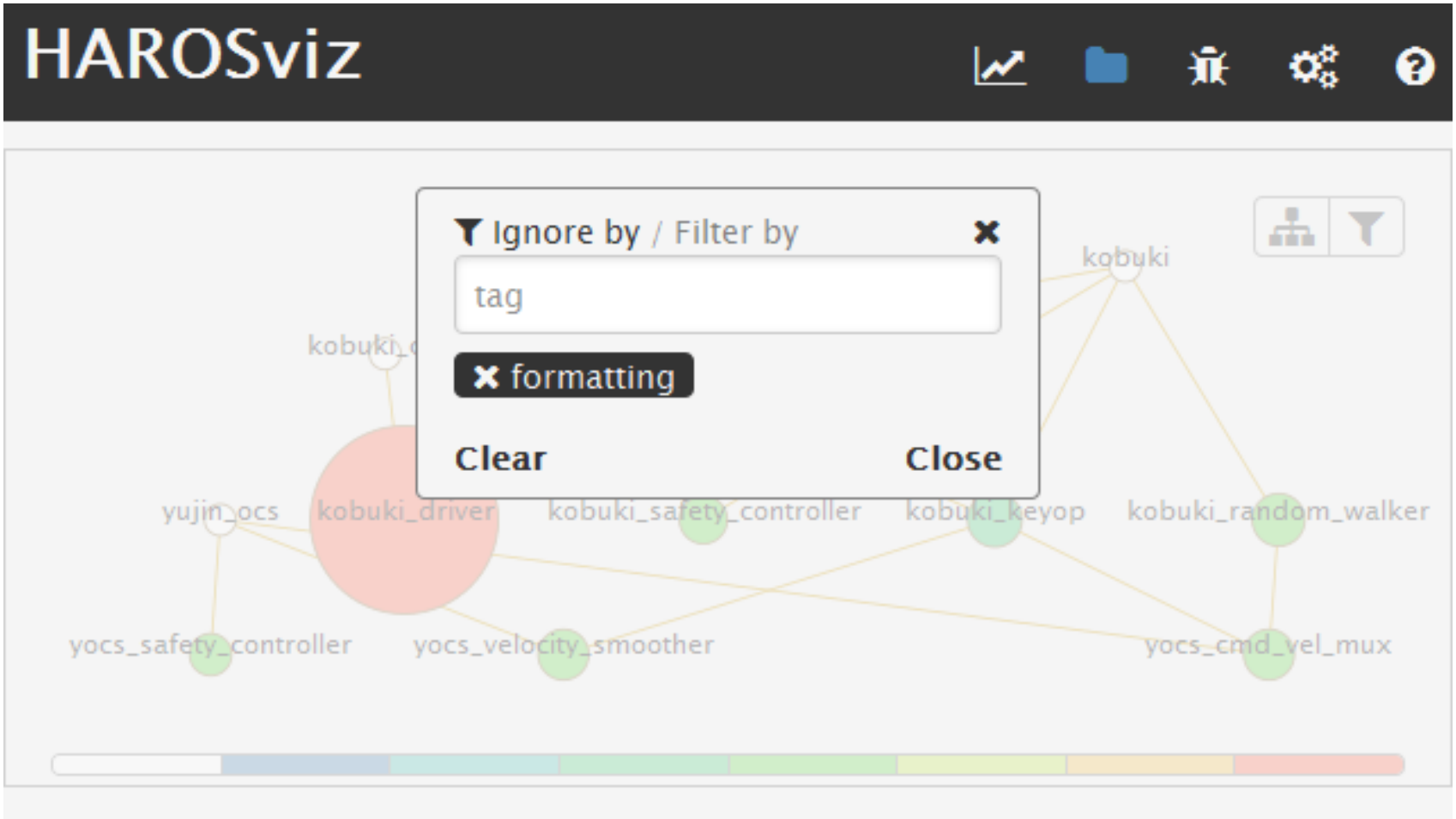
# The HAROS Framework: Visualisation

The visualiser builds a diagram of the analysed packages.
Package colours denote the amount of issues.



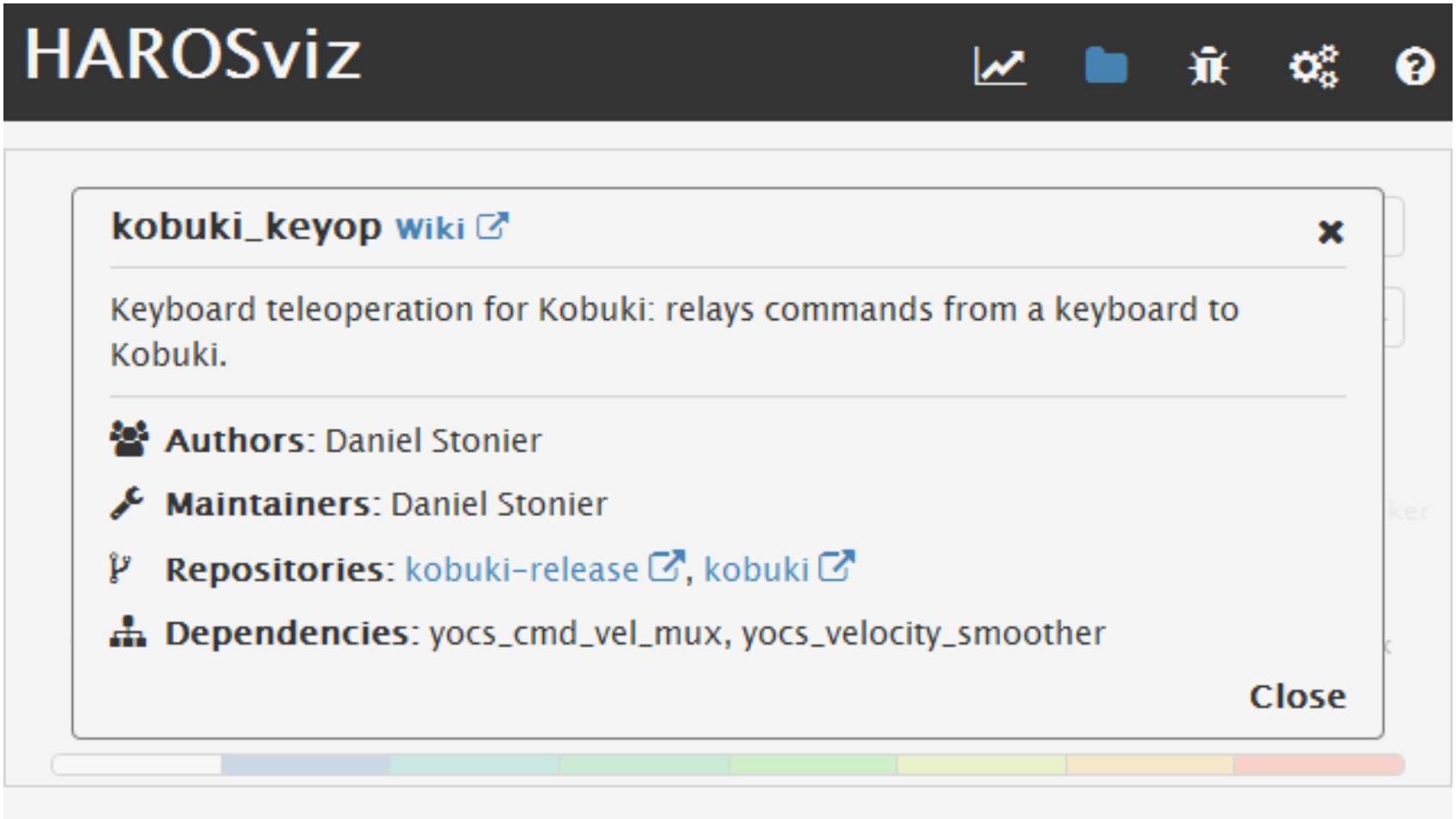
# The HAROS Framework: Visualisation

Issues can be filtered or ignored by tags.



# The HAROS Framework: Visualisation

Package details are also available.



# The HAROS Framework: Visualisation

Issues can be inspected in detail.

HAROSviz

Package kobuki_node Filter Page 26/27

**Issue #626** on slot_callbacks.cpp, line 267

**There shall be no unused variables.**

The function 'publishButtonEvent' is never used.

code-standards misra-cpp variables unused-variables

**Issue #627** on slot_callbacks.cpp, line 307

**There shall be no unused variables.**

The function 'publishCliffEvent' is never used.

code-standards misra-cpp variables unused-variables



HASLab HIGH-ASSURANCE SOFTWARE LABORATORY

# The HAROS Framework: Case Study

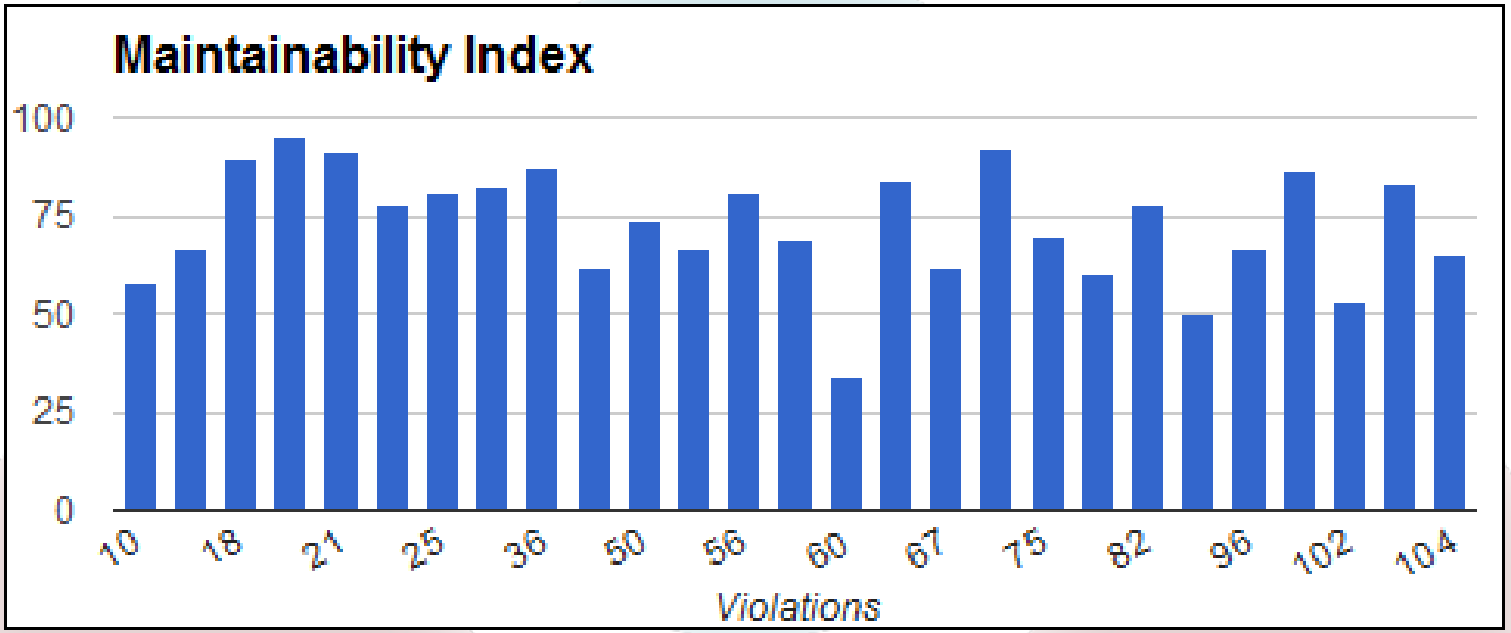
HAROS was applied on 11 ROS robots, using CCCC, Radon, Cpplint and Cppcheck as plug-ins.

- Analysis sample: 46 repositories – more than 350 000 lines of C++.
- Assessment of over 100 rules and 15 metrics.
  - Covering ROS and Google's C++ Style Guide, and a small portion of MISRA C++, HIC++, and JSF AV C++.
  - Source metrics: lines of code/comments, comment ratio, maintainability, dependencies, cyclomatic complexity, ...
  - Process metrics (from GitHub): commits, contributors, number of issues.
- Packages categorised as drivers, libraries, or applications.

# The HAROS Framework: Case Study

Some observations:

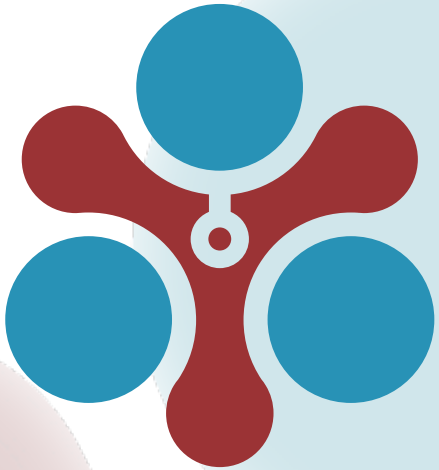
- The projects have thousands of coding rule violations.
- There are few correlations between metrics – the quality is inconsistent.
- Drivers and applications are more active – more developers and commits, but also more issues.



# Future Work

- Integration of stronger analysis techniques,
  e.g. formal verification and model checking.
- Model extraction from source code.
- Inter-operation between plug-ins.
- Integration with the catkin build system.
- Continuous tracking of package quality.

# Thank you!



HASLab

HIGH-ASSURANCE
SOFTWARE LABORATORY