



# HAROS: Quality Assessment of ROS Applications

Integration Workshop at IPA

André Santos

INESC TEC & Universidade do Minho, Portugal

January 31st, 2017



# Summary

- › Many interesting robotics applications (e.g. health, industry) require high levels of **safety** and **flexibility**.
- › In such scenarios, safety is usually ensured by **software**.
- › This kind of software must be **high-quality software**.
- › Our goal was to get a **panorama of the current quality level** in many popular ROS robots.
- › To achieve this goal, we built a **framework** to automate the collection of several **quality measures**.

# Software Quality

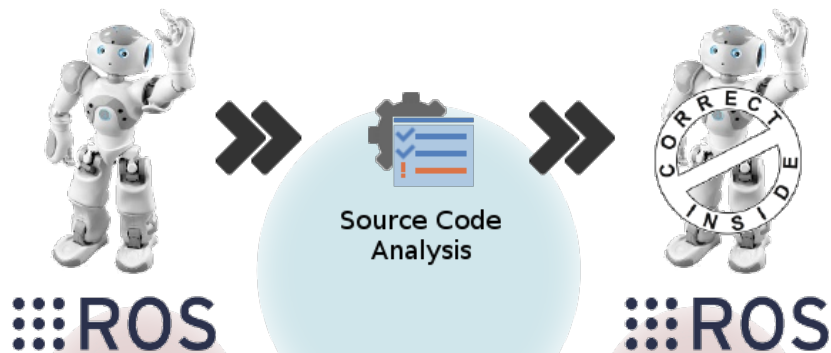
One way to minimise safety issues is to produce **high-quality software**.

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).

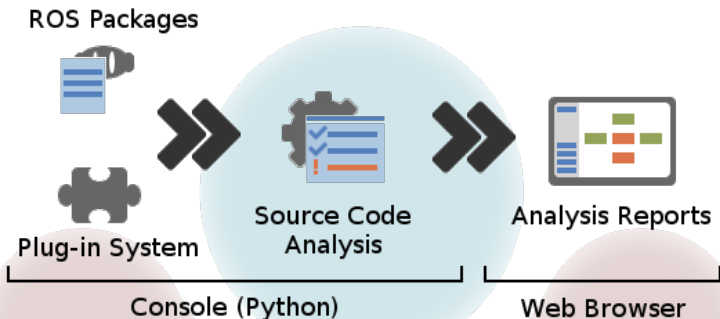
# The HAROS Framework: Overview

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



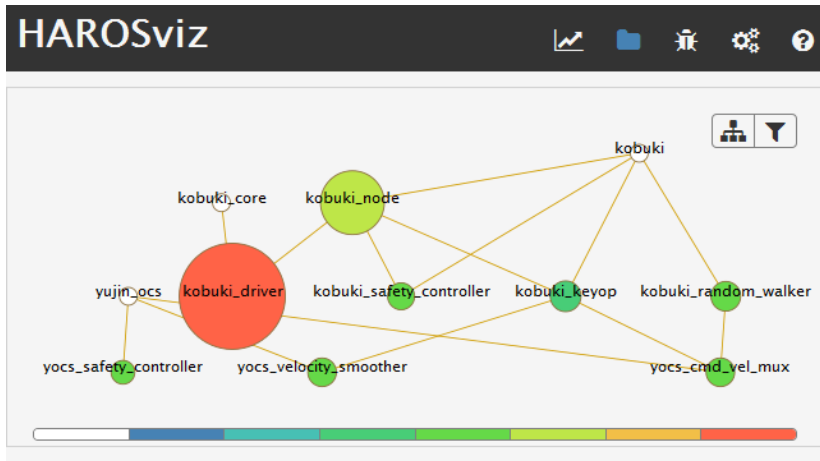
# The HAROS Framework: Main Features

- › **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.



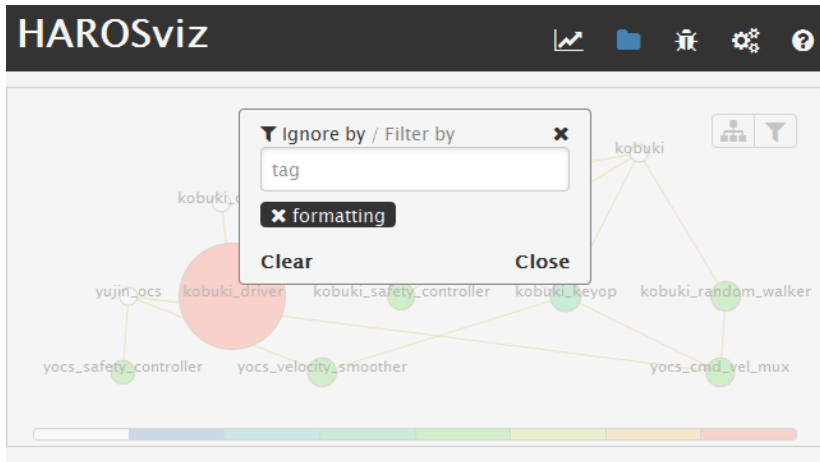
# The HAROS Framework: Visualisation

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



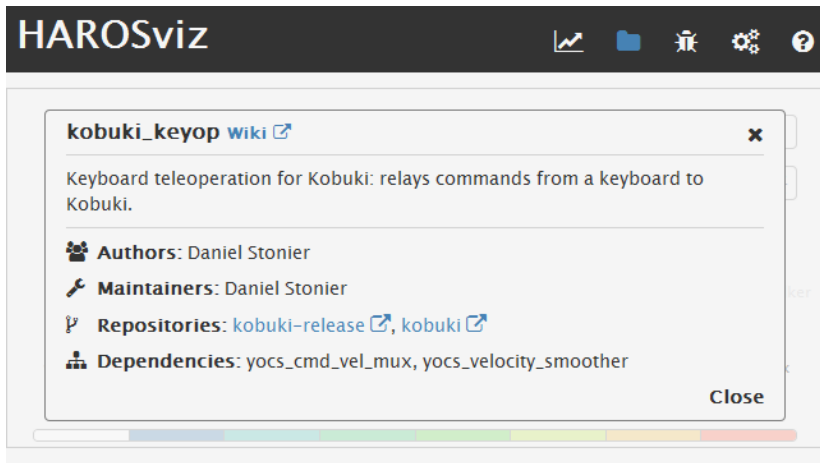
# The HAROS Framework: Visualisation

Issues can be filtered or ignored by tags.



# The HAROS Framework: Visualisation

Package details are also available.





The screenshot shows the HAROSviz application window. The title bar contains the text 'HAROSviz' and several icons: a line graph, a folder, a trash can, a gear, and a question mark. The main content area displays details for the 'kobuki\_keyop' package. At the top, the package name 'kobuki\_keyop' is followed by a 'Wiki' link and a close button. Below this is a description: 'Keyboard teleoperation for Kobuki: relays commands from a keyboard to Kobuki.' Further down, there are sections for 'Authors', 'Maintainers', 'Repositories', and 'Dependencies', each with an icon and text. A 'Close' button is located at the bottom right of the details panel. At the bottom of the application window, there is a horizontal progress bar with segments in white, blue, green, yellow, and red.


**HAROSviz**


**kobuki\_keyop** [Wiki](#) ✕

Keyboard teleoperation for Kobuki: relays commands from a keyboard to Kobuki.

 **Authors:** Daniel Stonier

 **Maintainers:** Daniel Stonier

 **Repositories:** [kobuki-release](#), [kobuki](#)

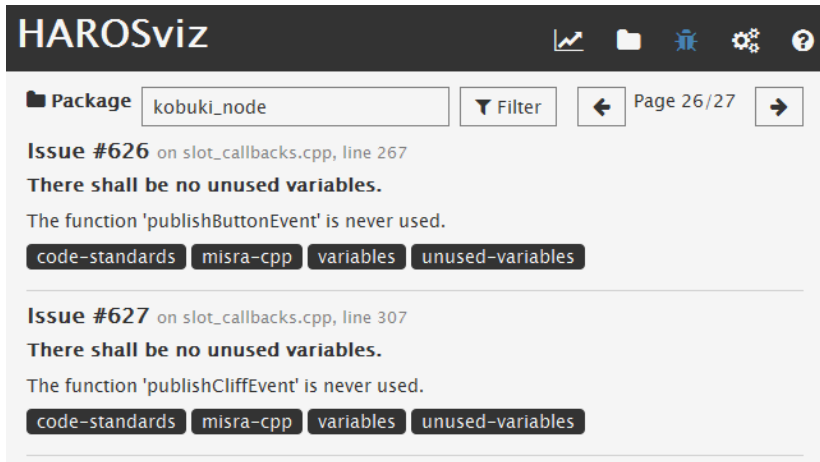
 **Dependencies:** yocs\_cmd\_vel\_mux, yocs\_velocity\_smoother

**Close**



# The HAROS Framework: Visualisation

Issues can be *inspected* in detail.



The screenshot shows the HAROSviz web interface. At the top, there is a dark header with the text "HAROSviz" on the left and several icons (line graph, folder, bug, gear, question mark) on the right. Below the header, there is a navigation bar with a "Package" dropdown menu set to "kobuki\_node", a "Filter" button, and a "Page 26/27" indicator with left and right arrows. The main content area displays two issues. Each issue starts with "Issue #626" or "Issue #627" followed by the file path and line number. The message for both is "There shall be no unused variables." Below each message is a sentence stating that a specific function is never used. At the bottom of each issue's description are four dark buttons with white text: "code-standards", "misra-cpp", "variables", and "unused-variables".

**HAROSviz**

Package:  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

# The HAROS Framework: Case Study

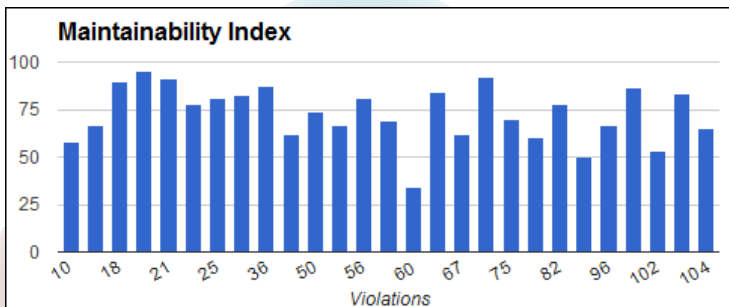
HAROS was applied on a few ROS robots, using **CCCC**, **Radon**, **Cpplint** and **Cppcheck** as plug-ins.

- › 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

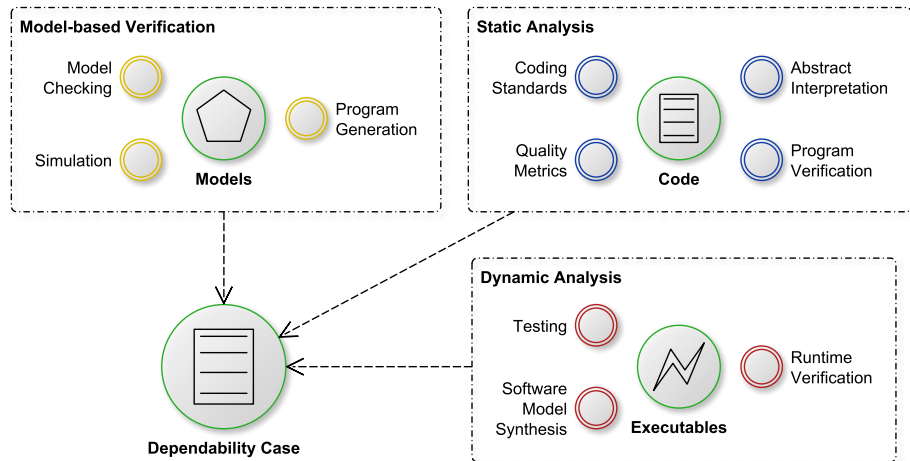
Some observations:

- › 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.



# What's Next: Software Safety, Better Quality

There are various techniques to assess software safety and quality.



# Research Questions

During our reserach, we hope to answer the following questions.

1. How to define a domain specific language for specifying safety properties over ROS code?
2. Is it possible to identify a catalogue of generic safety properties that apply to most ROS systems?
3. How to define model extraction techniques that reverse engineer ROS code into abstract models?
4. How to define ROS coding standards and best practices that make such model extraction feasible?

# HAROS: The NYI List

- › Ranking of issues (e.g. by **severity**).
- › Providing **ROS-specific metrics**.
- › Dedicated **view for raw metrics**.
- › **Continuous tracking** of package quality.
- › Integration with **catkin**.

Give HAROS a try at  
[git-afsantos.github.io/haros](https://git-afsantos.github.io/haros) or [github.com/git-afsantos/haros](https://github.com/git-afsantos/haros)

IROS 2016 paper at  
[haslab.uminho.pt/afsantos/publications](https://haslab.uminho.pt/afsantos/publications)

Reach me at  
[haslab.uminho.pt/afsantos](https://haslab.uminho.pt/afsantos)

Thank you!



**HASLab**  
HIGH-ASSURANCE  
SOFTWARE LABORATORY