“It is not sufficient to engineer
a solution.

The engineering team must make sure
that it is **the** correct solution.”
Rodrigues, G.

“It is not sufficient to engineer a solution.

The engineering team must make sure that it is **the** correct solution.”

How can

The engineering team make sure

that it is **the** correct solution

?

1) By design

2) Testing and Verification

1) By design

2) Testing and Verification 👀

Guidelines for Testing and Verifying robots in the field

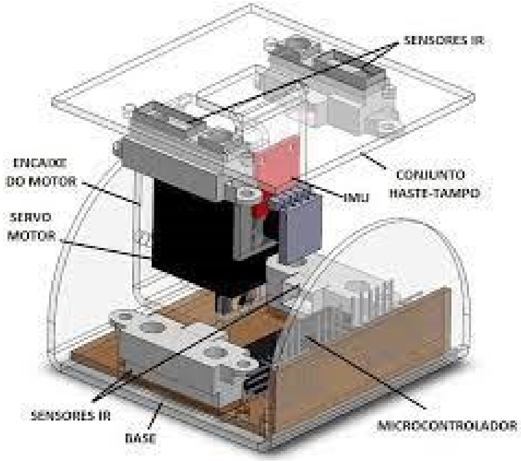
Ricardo D. Caldas
Brasília, Brasil, Feb. 2024

CHALMERS
UNIVERSITY OF TECHNOLOGY

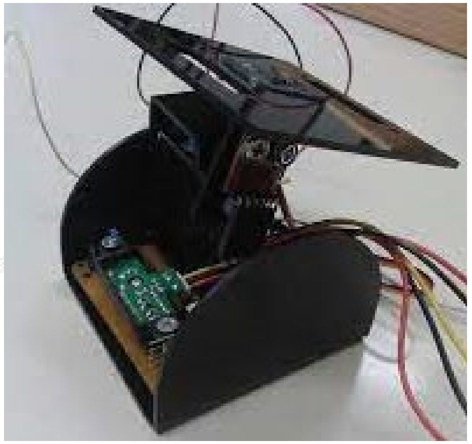
WASP
WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

About me

- Control and Automation Engineer (UnB, Brazil)
- Masters in Dependability and SE (UnB, Brazil)
- PhD student, Robotics SE (Chalmers, Sweden)

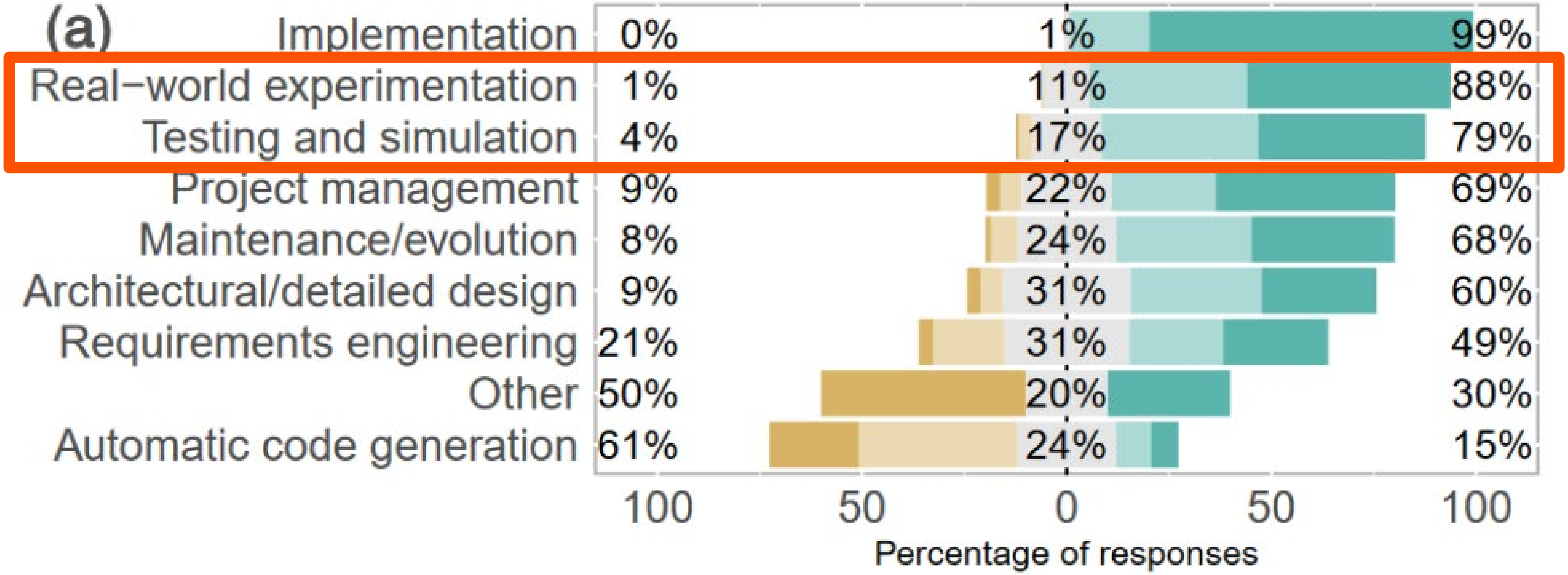


2014



🔥 On what software engineering activities do roboticists spend most of their time?

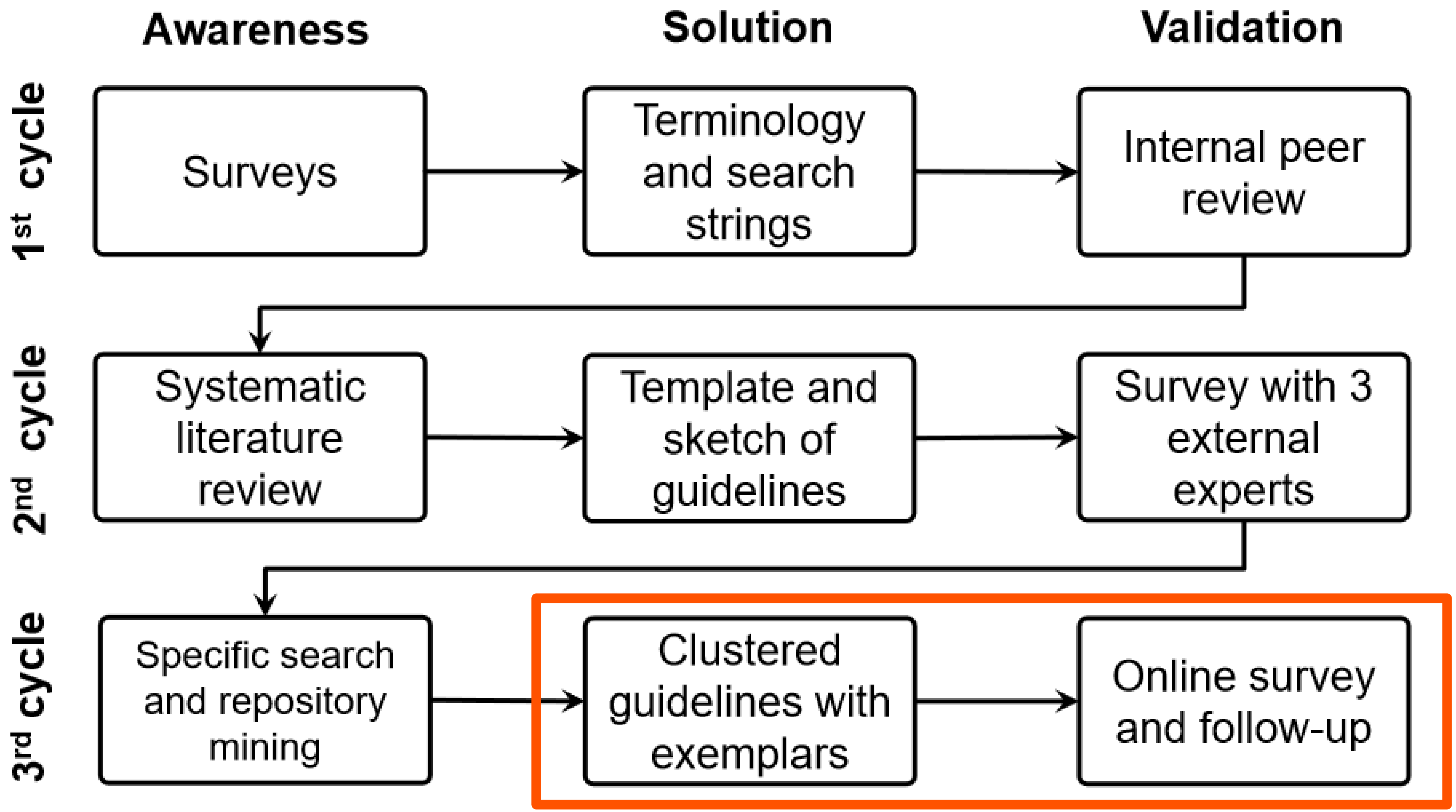
156 respondents



Practitioners mainly focus on implementation and real-world experimentation (preferred over simulation) during software development, typically following agile-style processes.

IT IS UNCLEAR HOW ROS SUPPORTS SYSTEMATIC RUNTIME VERIFICATION AND FIELD-BASED TESTING.

How does ROS supports RV and FbT?



Awareness

Solution

Validation

1st cycle

Surveys

Terminology and search strings

Internal peer review

2nd cycle

Systematic literature review

Template and sketch of guidelines

Survey with 3 external experts

3rd cycle

Specific search and repository mining

Clustered guidelines with exemplars

Online survey and follow-up

Overview of the guidelines to Field-based Testing (FT) and Ru ems

Legend

Prepari
& Runti
based sy

ems

CI1. Identify
constraints

Code d
fo

CD1. Strive
with single r

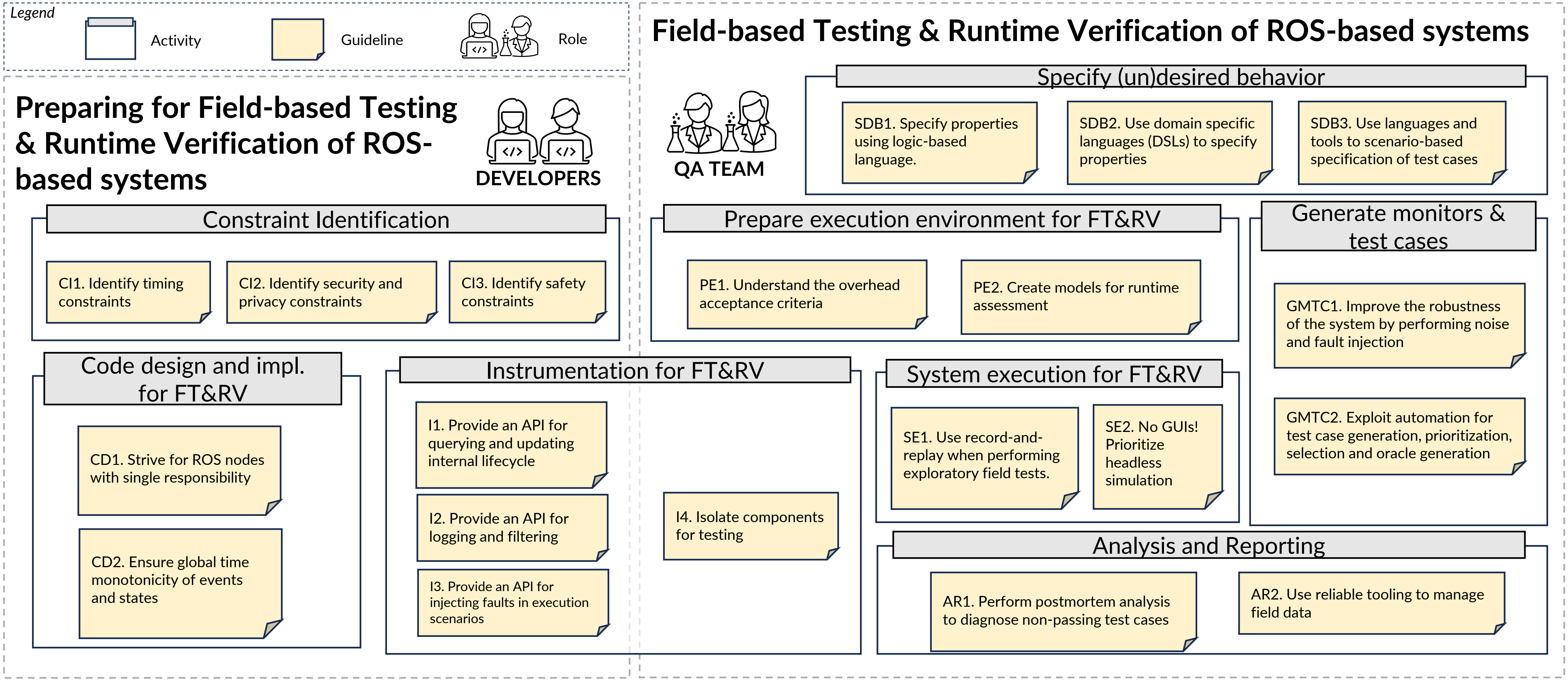
CD2. Ensure g
monotonicity c
and states

e

GUIDELINES OVERVIEW

https://ros-rvft.github.io/

20 guidelines | 8 clusters | 8 for Devs + 12 for RV and FT



Legend

Activity

Guideline

Role

Preparing for Field-based Testing & Runtime Verification of ROS-based systems

DEVELOPERS

Constraint Identification

CI1. Identify timing constraints

CI2. Identify security and privacy constraints

CI3. Identify safety constraints

Code design and impl. for FT&RV

CD1. Strive for ROS nodes with single responsibility

CD2. Ensure global time monotonicity of events and states

Instrumentation for FT&RV

I1. Provide an API for querying and updating internal lifecycle

I2. Provide an API for logging and filtering

I3. Provide an API for injecting faults in execution scenarios

I4. Isolate components for testing

Field-based Testing & Runtime Verification of ROS-based systems

QA TEAM

Specify (un)desired behavior

SDB1. Specify properties using logic-based language.

SDB2. Use domain specific languages (DSLs) to specify properties

SDB3. Use languages and tools to scenario-based specification of test cases

Prepare execution environment for FT&RV

PE1. Understand the overhead acceptance criteria

PE2. Create models for runtime assessment

Generate monitors & test cases

GMTC1. Improve the robustness of the system by performing noise and fault injection

GMTC2. Exploit automation for test case generation, prioritization, selection and oracle generation

System execution for FT&RV

SE1. Use record-and-replay when performing exploratory field tests.

SE2. No GUIs! Prioritize headless simulation

Analysis and Reporting

AR1. Perform postmortem analysis to diagnose non-passing test cases

AR2. Use reliable tooling to manage field data

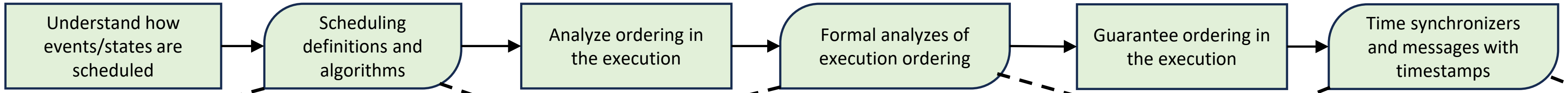
CD2. Ensure global time monotonicity of events and states

Non-determinism in the scheduling of events can lead to unexpected behavior, compromising the reliability of tests and hindering their reproduction.

“The development team should ensure global time monotonicity of events and states to avoid potential scheduling non-determinism”

Ensuring global time monotonicity of events and states permits to address the potential non-determinism in the scheduling of events in ROS-based applications

PROCESS:



Understand how events/states are scheduled → Scheduling definitions and algorithms → Analyze ordering in the execution → Formal analyzes of execution ordering → Guarantee ordering in the execution → Time synchronizers and messages with timestamps

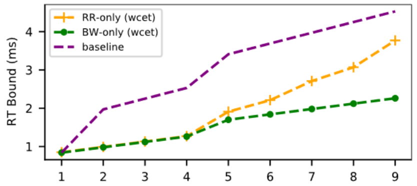
Examples of scheduling in ROS 2

“ROS 2 schedules callbacks using a non-preemptive algorithm that consumes messages depending on their type. Unlike typical real-time priority-based scheduling algorithms, ROS 2 does not execute callbacks in their activation instances. Thus, non-time-based messages are scheduled in a round-robin fashion ” Chaaban, K. in A New Algorithm for Real-Time Scheduling and Resource Mapping for Robot Operating Systems (ROS). Applied Sciences 13.3 (2023)

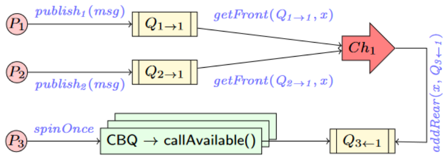
Choi, H., et al. "PiCAS: New design of priority-driven chain-aware scheduling for ROS2." *RTAS*. IEEE, 2021.

https://github.com/rtenlab/ros2-picas

Examples of ordering analyzes on ROS applications



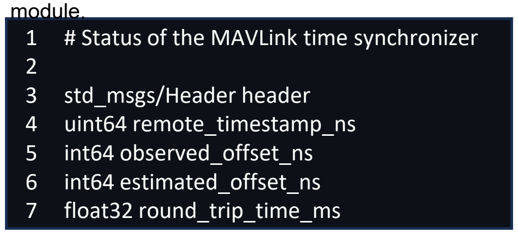
Blaß, T., et al. "A ROS 2 response-time analysis exploiting starvation freedom and execution-time variance." *RTSS*. IEEE, 2021.



Halder, R., et al. "Formal verification of ROS-based robotic applications using timed-automata." *FormaliSE*. IEEE, 2017.

Example of message with timestamps

Annotate ROS messages with timestamps for instance the MAVLink time synchronizer module.



```
# Status of the MAVLink time synchronizer

std_msgs/Header header
uint64 remote_timestamp_ns
int64 observed_offset_ns
int64 estimated_offset_ns
float32 round_trip_time_ms
```

https://github.com/mavlink/mavros/blob/ros2/mavros_msgs/msg/TimesyncStatus.msg

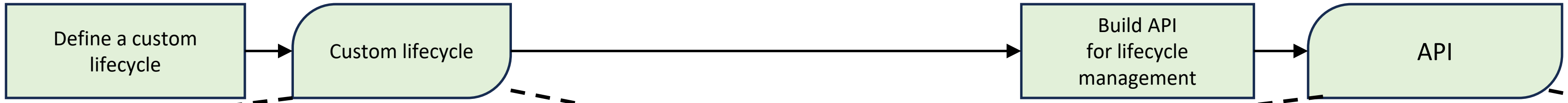
I1. Provide an API for querying and updating internal lifecycle

In ROS, internal states are typically hidden limiting the ability to diagnose and understand unpredicted behavior.

“To facilitate field-based testing, the development team should adopt custom lifecycle conventions and prepare an API for querying and updating the internal life-cycle.”

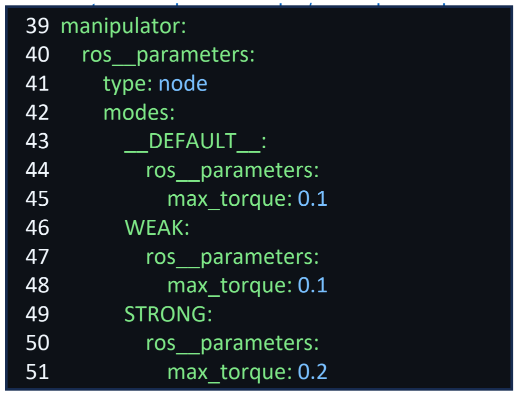
ROS nodes with lifecycle management provide:
(1). structured way to manage nodes and interactions;
(2). ensuring the right state for testing;
(3). helps mitigate dangling nodes that are not in use;

PROCESS:



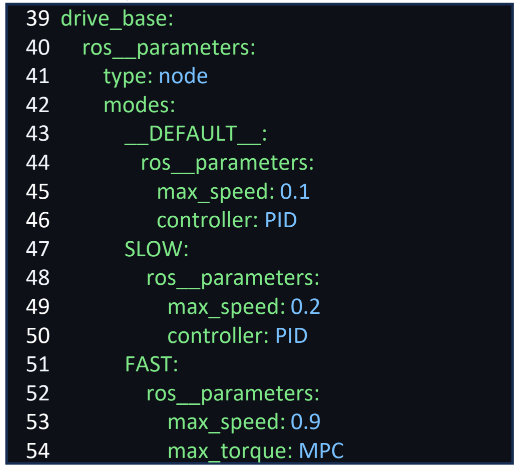
Examples on defining a custom lifecycle

https://github.com/micro-ROS/system_modes/.../



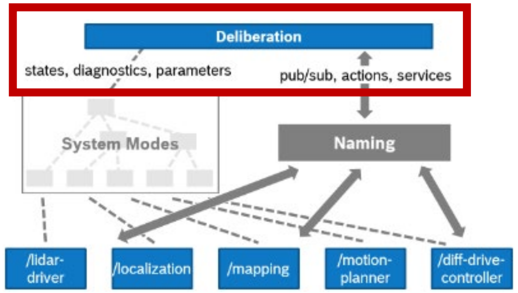
```
manipulator:
  ros__parameters:
    type: node
    modes:
      __DEFAULT__:
        ros__parameters:
          max_torque: 0.1
      WEAK:
        ros__parameters:
          max_torque: 0.1
      STRONG:
        ros__parameters:
          max_torque: 0.2
```

https://github.com/micro-ROS/system_modes/.../



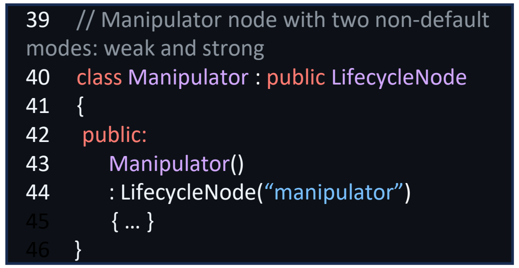
```
drive_base:
  ros__parameters:
    type: node
    modes:
      __DEFAULT__:
        ros__parameters:
          max_speed: 0.1
          controller: PID
      SLOW:
        ros__parameters:
          max_speed: 0.2
          controller: PID
      FAST:
        ros__parameters:
          max_speed: 0.9
          max_torque: MPC
```

Example on Lifecycle Management



Nordmann, A., et al. “System modes-digestible system
(re-) configuration for robotics.” 2021 IEEE/ACM 3rd RoSE.

https://github.com/micro-ROS/system_modes/.../system_modes_examples/manipulator.cpp



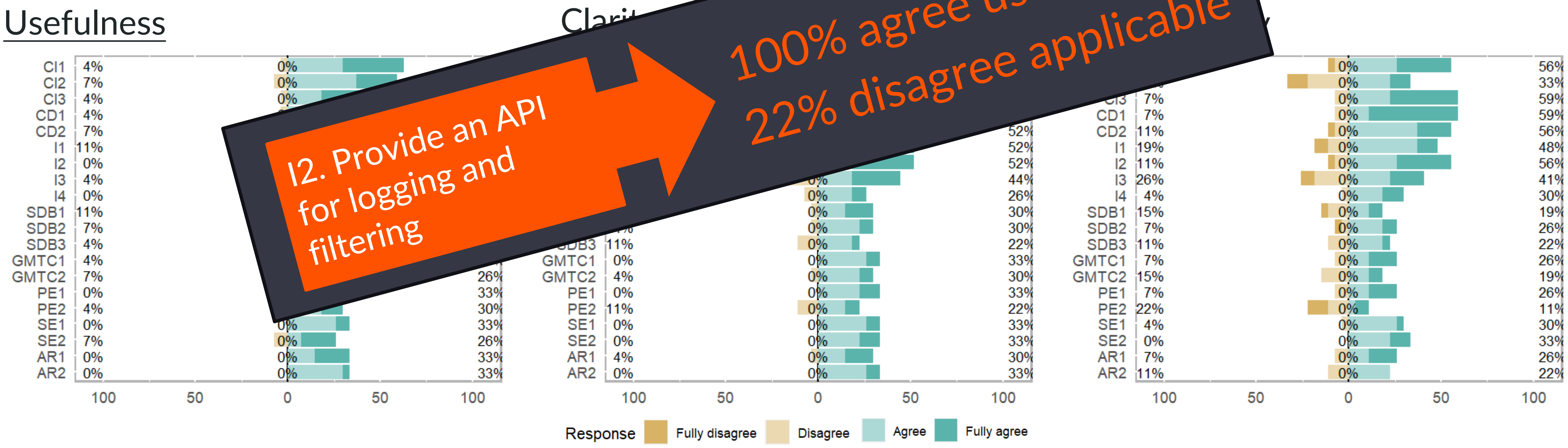
```
// Manipulator node with two non-default modes: weak and strong
class Manipulator : public LifecycleNode
{
 public:
   Manipulator()
   : LifecycleNode(“manipulator”)
   { … }
}
```

cool, but so what?

are these guidelines
useful, clear, applicable?

How do developers and QA teams like our guidelines?

- 55 responses (Industry and Academics)
- Service robotics, marine robotics and industrial automation
- 22% (1 – 3 yrs), 22% (3 – 5 yrs) and 40% (> 10 yrs)



I2. Provide an API for logging and filtering

100% agree useful

22% disagree applicable

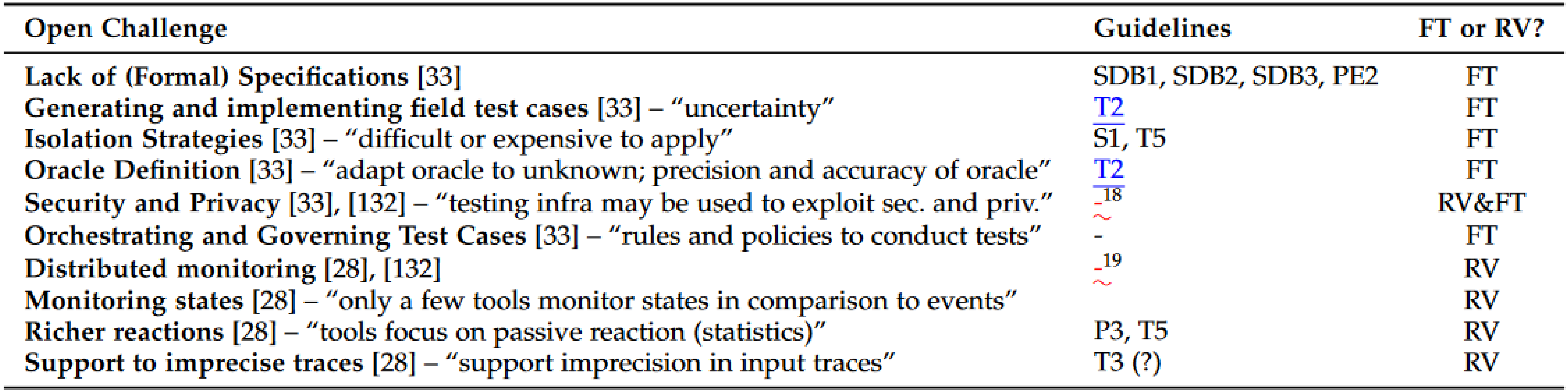
Future Work (under construction…)

- Guidelines that never made it to the end

  “Explicitly annotate ROS nodes with contracts”

  “Use Closed-Form Expressions for Recording Time-Continuous Traces”

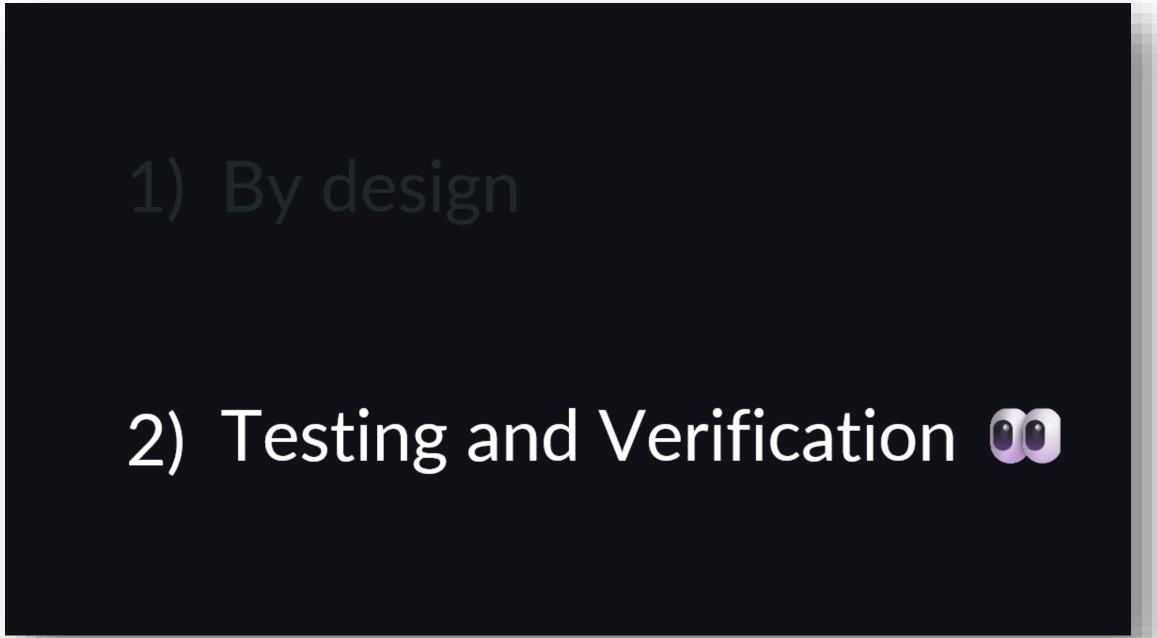
- How do guidelines address the state-of-the-art of field-based testing and runtime verification?



Open Challenge	Guidelines	FT or RV?
Lack of (Formal) Specifications [33]	SDB1, SDB2, SDB3, PE2	FT
Generating and implementing field test cases [33] – “uncertainty”	T2	FT
Isolation Strategies [33] – “difficult or expensive to apply”	S1, T5	FT
Oracle Definition [33] – “adapt oracle to unknown; precision and accuracy of oracle”	T2	FT
Security and Privacy [33], [132] – “testing infra may be used to exploit sec. and priv.”	-[18]	RV&FT
Orchestrating and Governing Test Cases [33] – “rules and policies to conduct tests”	-	FT
Distributed monitoring [28], [132]	-[19]	RV
Monitoring states [28] – “only a few tools monitor states in comparison to events”		RV
Richer reactions [28] – “tools focus on passive reaction (statistics)”	P3, T5	RV
Support to imprecise traces [28] – “support imprecision in input traces”	T3 (?)	RV

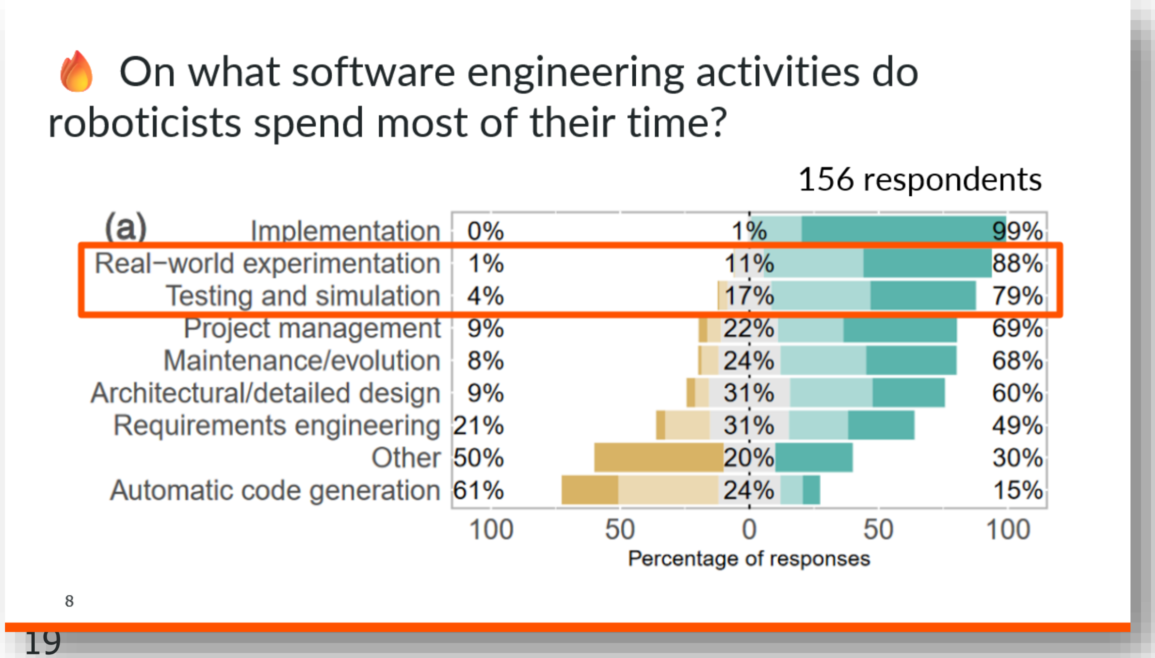
Take aways

👀 **Real-world** Testing and Verification help to engineer **the** correct solution;
🔥 ROS does not provide extensive support to real-world testing;
🤓 Mixed-methods (SLR + repo mining) are a way to provide actionable results.



1) By design

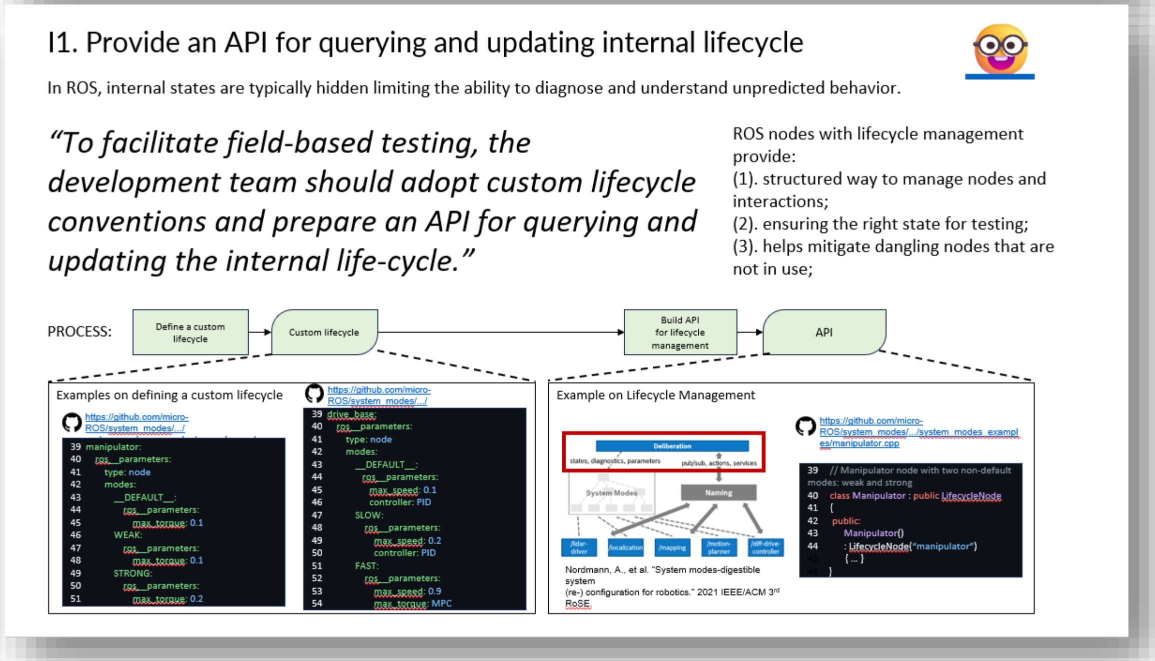
2) Testing and Verification 👀



🔥 On what software engineering activities do roboticists spend most of their time?

156 respondents

8



I1. Provide an API for querying and updating internal lifecycle 🤓

In ROS, internal states are typically hidden limiting the ability to diagnose and understand unpredicted behavior.

"To facilitate field-based testing, the development team should adopt custom lifecycle conventions and prepare an API for querying and updating the internal life-cycle."

ROS nodes with lifecycle management provide:
(1). structured way to manage nodes and interactions;
(2). ensuring the right state for testing;
(3). helps mitigate dangling nodes that are not in use;

Examples on defining a custom lifecycle

https://github.com/micro-ROS/system_modes/.../

```
manipulator:
  ros__parameters:
    type: node
    modes:
      __DEFAULT__:
        ros__parameters:
          max_torque: 0.1
      WEAK:
        ros__parameters:
          max_torque: 0.1
      STRONG:
        ros__parameters:
          max_torque: 0.2
```

https://github.com/micro-ROS/system_modes/.../

```
drive_base:
  ros__parameters:
    type: node
    modes:
      __DEFAULT__:
        ros__parameters:
          max_speed: 0.1
          controller: PID
      SLOW:
        ros__parameters:
          max_speed: 0.2
          controller: PID
      FAST:
        ros__parameters:
          max_speed: 0.9
          max_torque: MPC
```

Example on Lifecycle Management

Nordmann, A., et al. "System modes-digestible system (re-) configuration for robotics." 2021 IEEE/ACM 3rd RoSE.

https://github.com/micro-ROS/system_modes/.../system_modes_examples/manipulator.cpp

```
// Manipulator node with two non-default modes: weak and strong
class Manipulator : public LifecycleNode
{
 public:
    Manipulator()
    : LifecycleNode("manipulator")
    { ... }
}
```

Thanks!

ricardo.caldas@chalmers.se
https://rdinizcal.github.io

Check the guidelines :

https://ros-rvft.github.io/

CHALMERS
UNIVERSITY OF TECHNOLOGY

WASP
WALLENBERG AI,
AUTONOMOUS SYSTEMS
AND SOFTWARE PROGRAM