Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

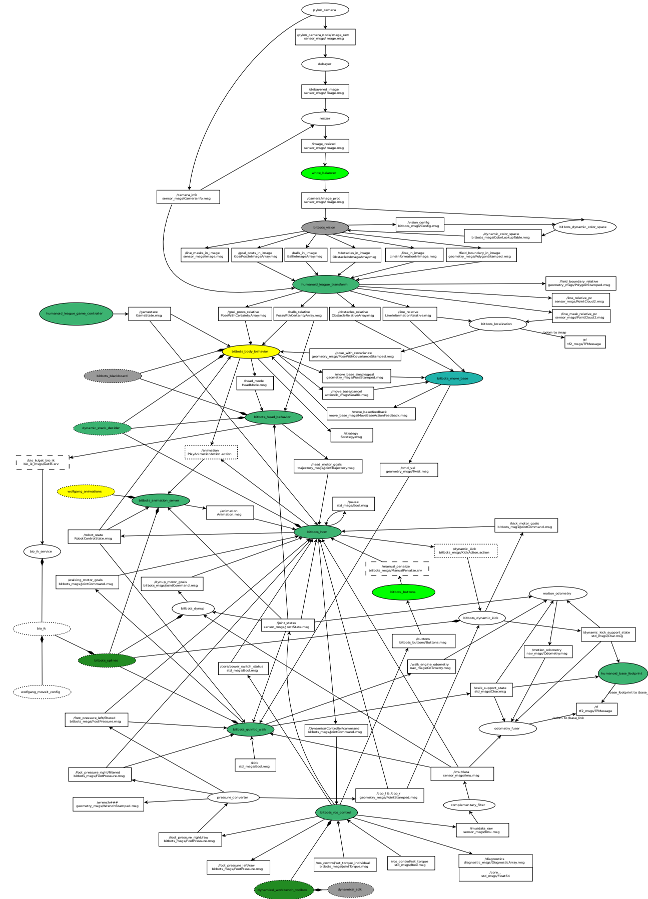
Finn-Thorben Sell

# Integration of Software Development Testing Practices in Robotics

# Hamburg Bit-Bots and RoboCup

# Motivation

- Complex software
- Much time spent testing
- Many people involved



# Background and Requirements

- Environment
  - ROS *(Robot Operating System)*
  - Python
  - C++
- System tests
- Component tests
- Automation
- Unified interface

# Related Work

2018 IEEE 11th International Conference on Software Testing, Verification and Validation

## Crashing simulated planes is cheap: Can simulation detect robotics bugs early?

Christopher Steven Timperley*, Afsoon Afzal*, Deborah S. Katz*, Jam Marcos Hernandez† and Claire Le Goues*
*Carnegie Mellon University, Pittsburgh, PA
†State University of New York at Potsdam, Potsdam, NY
Email: ctimperley@cmu.edu, afsoona@cs.cmu.edu, dskatz@cs.cmu.edu, jamarck96@gmail.com, clegoues@cs.cmu.edu

***Abstract*—Robotics and autonomy systems are becoming increasingly important, moving from specialised factory domains to increasingly general and consumer-focused applications. As such systems grow ubiquitous, there is a commensurate need to protect against potentially catastrophic harm. System-level testing in simulation is a particularly promising approach for assuring robotics systems, allowing for more extensive testing in realistic scenarios and seeking bugs that may not manifest at the unit-level. Ideally, such testing could find critical bugs well before expensive field-testing is required. However, simulations can only model coarse environmental abstractions, contributing to a common perception that robotics bugs can only be found in live deployment. To address this gap, we conduct an empirical study on bugs that have been fixed in the widely used, open-source ARDUPILOT system. We identify bug-fixing commits by exploiting commenting conventions in the version-control history. We provide a quantitative and qualitative evaluation of the bugs, focusing on characterising how the bugs are triggered and how they can be detected, with a goal of identifying how they can be best identified in simulation, well before field testing. To our surprise, we find that the majority of bugs manifest under simple conditions that can be easily reproduced in software-based simulation. Conversely, we find that system configurations and forms of input play an important role in triggering bugs. We use these results to inform a novel framework for testing for these and other bugs in simulation, consistently and reproducibly. These contributions can inform the construction of techniques for automated testing of robotics systems, with the goal of finding bugs early and cheaply, without incurring the costs of physically testing for bugs in live systems.**

***Index Terms*—automated testing, empirical study, robotics, autonomous vehicles, dataset, repository mining, ARDUPILOT**

However, as safety-critical systems, failures in robotics systems can be expensive and, in some cases, deadly. As potentially dangerous robotics and autonomous systems increasingly come into contact with humans, it is essential to develop effective quality-assurance methods. Field testing, unit testing, and verification remain important to quality assurance in robots, but they cannot cover all situations a system may potentially encounter. Field testing is especially critical in safety-critical systems and can identify key issues, but failures at this late stage can be enormously expensive. One notable example of the need for simulation testing is the ExoMars Lander, which crashed in October 2016 at an approximate cost of $350 million in materials and time. After the crash, investigators were able to recreate the circumstances of the crash in simulation, which led to the simulated vehicle also crashing (exo, 2016).

Instead, ideally, bugs can be identified as early as possible, reducing the cost of finding and fixing them, and before they have manifested in physical systems (Williamson, 2008).

Automated full-system testing (e.g., Liu and Mei 2014) in simulation will ideally assist in addressing these deficiencies. Indeed, this is our long-term research ambition: to produce highly effective techniques for automatically detecting bugs in real-world robotic systems through the use of software-based simulation, dramatically reducing the cost of such bugs by avoiding the need of costly deployment. However, simulation, by necessity, represents a simplified abstraction of

focus

software testing

## A Survey of Unit Testing Practices

**Per Runeson,** *Lund University*

Companies participated in a survey to define unit testing and evaluate their strengths and weaknesses at applying it. Others can use the survey to judge and improve their own practices.

Unit testing is testing of individual units or groups of related units."[1] You know the definition by the book, but what does it mean to you? What are a company's typical strengths and weaknesses when applying unit testing? Per Beremark and I surveyed unit testing practices on the basis of focus group discussions in a software process improvement network (SPIN) and launched a questionnaire to validate the results. I aimed to go beyond standard terminology definitions and investigate what practitioners refer to when they talk about unit testing. Based on this common understanding, I also investigated unit testing practices' strengths and weaknesses.

The survey revealed a consistent view of unit testing's scope, but participants didn't agree on whether the test environment is an isolated harness or a partial software system. Furthermore, unit testing is clearly a developer issue, both practically and strategically. Neither test management nor quality management seem to impact unit testing strategies or practices. Unit tests are structural, or white-box

### The survey

SPIN-syd is a noncommercial network focused on software process improvement issues. It comprises representatives from 50 companies with software as a major part of their business. The companies range from consultancy firms with one employee to regional branches of multinational companies with hundreds of developers. The network represents various application domains with a focus on embedded systems. Lund University researchers and PhD students also belong to the network. The network has a monthly three-hour meeting and oc-

## A Generic Testing Framework for Test Driven Development of Robotic Systems

Ali Paikan, Silvio Traversaro, Francesco Nori, and Lorenzo Natale
Istituto Italiano di Tecnologia (IIT), Genova, Italy
{ali.paikan,silvio.traversaro,francesco.nori,lorenzo.natale}@iit.it

**Abstract.** This paper proposes a generic framework for test driven development of robotic systems. The framework provides functionalities for developing and running unit tests in a language and middleware independent manner. Tests are developed as independent plug-ins to be loaded and executed by an automated tool. Moreover, a fixture manager prepares the setup (e.g., running robot drivers or simulator) and actively monitors that all the required resources are available before and during the execution of the tests. These functionalities effectively accelerate the development process and cover different levels of robotic system testing. The paper describes the framework and provides realistic examples to show how it has been used to support software development on our robotic platform.

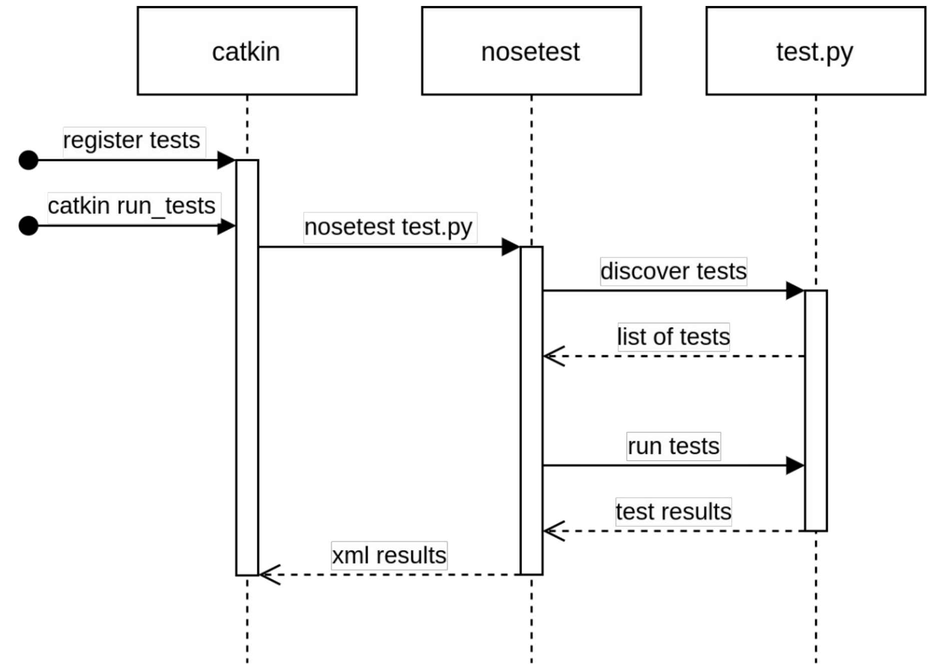
**Keywords:** Robot testing framework, Unit testing, Test-driven development, Software engineering, Robotics

### 1 Introduction

Autonomous robots have evolved in complex systems that are increasingly difficult to engineer and develop. A possible approach to tame such complexity is to divide the system into simpler units that are independently developed, tested and integrated at a later stage. Further testing is consequently performed on the whole system; this may trigger re-development or debugging of the individual components in an iterative process.

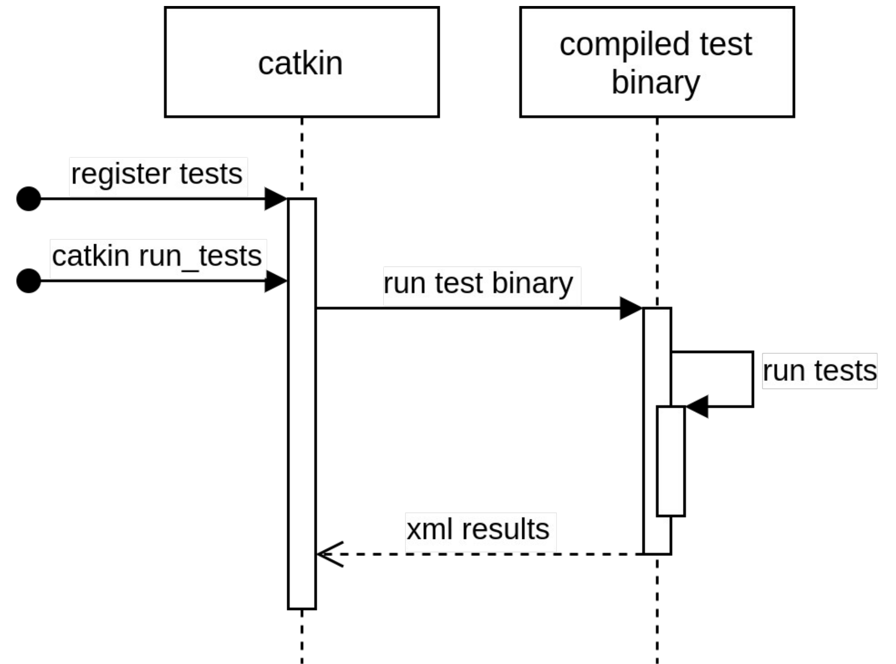
# Choosing a Testing Framework (Python)

- *nosetest* for Python
  - ROS support
  - Extension of unittest
  - Auto-Discovery
  - Established but discontinued
  - Minimal when writing tests



# Choosing a Testing Framework (C++)

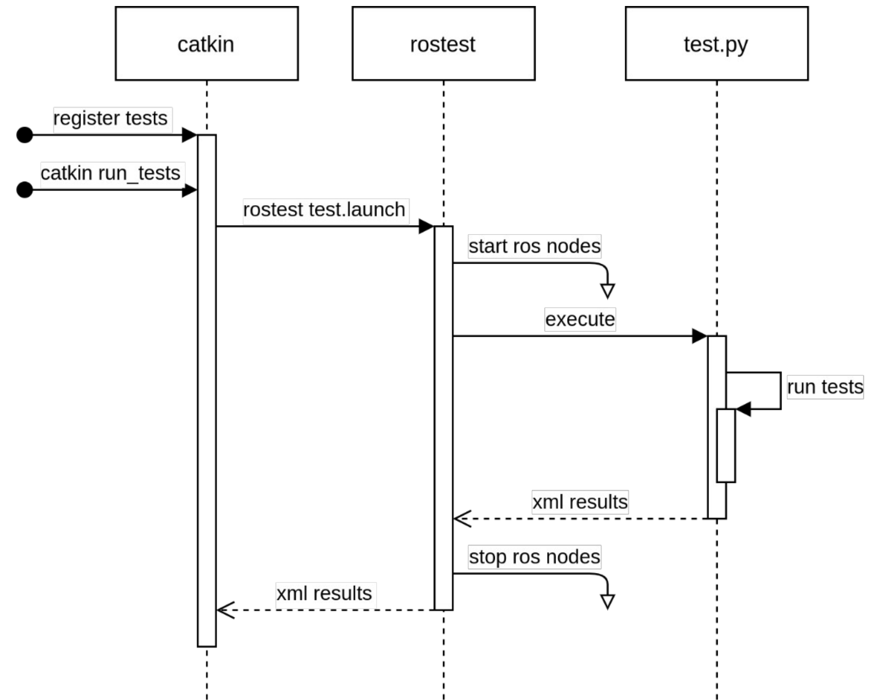
- *GoogleTest* for C++
  - ROS support
  - Popular
  - Supports many use cases



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

# Choosing a Testing Framework (System Tests)

- *rostest* for System-Tests
  - ROS support
  - Launch-File with <test/> tag
  - Minimal when writing tests



# Requirements?

- Environment ✓
- System tests ~
- Component tests ~
- Automation
- Unified interface ✓

# ***bitbots_test*** **Library**

- Auto-Discovery for C++ and rostests
- Test classification
- New assertions
- Common utilities

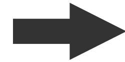
# Requirements?

- Environment ✓
- System tests ✓
- Component tests ✓
- Automation
- Unified interface ✓

# Build Automation Platform

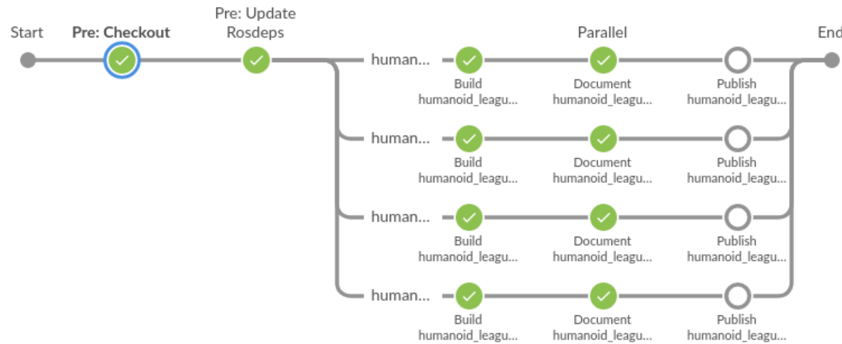
Requirements

- Support our environment
- Jobs as config
- On-Premise runners
- Low budget



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

# Build Automation in Action



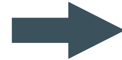

- Jenkins
- *bitbots_jenkins_library*
- *rosdep* definitions for our packages

# Requirements?

- Environment ✓
- System tests ✓
- Component tests ✓
- Automation ✓
- Unified interface ✓

# Qualitative Evaluation

- Testing Event to write tests
  - Guided Introduction
  - Documentation
  - Accompanying Survey



Testing will be easier

Quality will improve

Participants would like to use *bitbots_test*

Reproducibility issues

Debugging tests is hard

## Ongoing Work

- Improve Reproducibility
- Quantitative analysis

## Future Work

- Implement interactive tests
- Integrate *pytest* and *nose2*
- Generalize *bitbots_jenkins_library*
- Upstream parts of *bitbots_test*
- Nightly Tests
- Linting

# Conclusion

- Abstraction and utilities with *bitbots_test*
- Automation with Jenkins
- Tied into ROS and GitHub