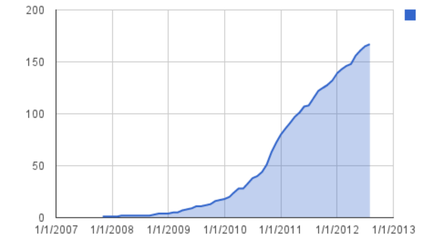
discovering problems

What are barriers to efficient ROS debugging?

motivation

The Robot Operating System, or ROS, is a leading robotics framework in academic research, and has also recently seen rapidly increased adoption in industrial settings [fig 1]. A framework is a set of reusable modules that requires its client plugins to conform to a predefined architecture. Thus, frameworks impose constraints on client plugins to ensure predictability, compatibility and code reusability. Some of these constraints are expressed as directives: statements which succinctly warn against improper operation [fig 2]. In conjunction with a wider project studying framework usability, we have studied and characterized the particular barriers programmers face when completing program repair tasks in the ROS framework.



1 ROS now has more than 160 publicly released and indexed software repositories

method

1. Find directives in the Robot Operating System Wiki, Q/A sites, forums, textbooks and tutorials
2. Build study systems which rely on correctly implemented directives
3. Modify our systems to contravene these directives, thus breaking our systems
4. Watch ROS programmers with varying levels of experience fix these broken test systems
5. Record their vocalized thoughts, screens and answers during a debriefing interview
6. Manually generate a detailed chronological repair summary
7. Apply a coding frame [fig 3] to capture and categorize this large amount of qualitative data
8. Analyze resulting groups of data to extract problems generalizable to all or most programmers

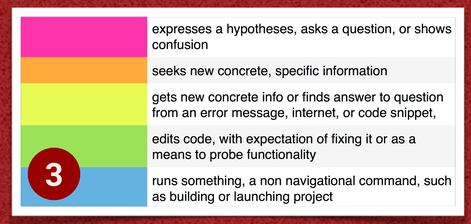
> “Omitting ros::Spin() will likely cause your program to exit shortly after it starts.”

2

results

Participants faced significant difficulty understanding and visualizing the general structure of unfamiliar ROS systems. In particular, they found it challenging to:

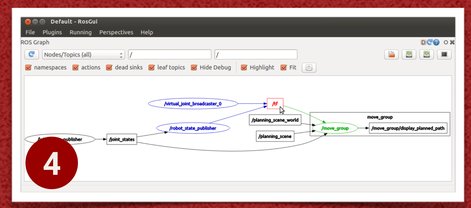
- find source files which correspond to particular abstract ROS entities (nodes, services, topics)
- find where a given node was initialized
- understand and visualize ROS service calls
- discern which files were executed in a given launch
- find out which service calls a node has made
- find the contents of service call requests and responses
- know which nodes have been active since the beginning of execution.



3

future questions

How are the challenges encountered in ROS debugging different to those encountered in debugging other software frameworks, and to those encountered in debugging in general?
Can we improve visualization tools [fig 4] to remedy the problems ROS developers face?
Can we automatically and effectively enforce ontologically encoded directives?



4

team

David Widder dwidder@uoregon.edu
Claire Le Goues clegoues@cs.cmu.edu Joshua Sunshine sunshine@cs.cmu.edu

Carnegie Mellon University | UNIVERSITY OF OREGON