Name

Submitting patches to core packages through maintainers

Context

An application developer discovers a bug in a core ROS package (e.g. roscpp), corrects it and tests the patch on his own computer. The fix should be merged into the main code base, but that should be done in a controlled manner.

Problem

How to ensure quality of the proposed changes to the core – and of the software as a whole after integration of those changes – and at the same time encourage newcomers and members of the ROS community to participate in the development?

Forces

Striving for quality

Everybody and especially maintainers are interested in maintaining the quality of core packages, as the introduction of new bugs or regressions or reducing maintainability will affect all current and future users of ROS. (Perceived) usability and stability have a direct influence on the reputation of ROS – especially in industrial settings where quality of the core packages is often considered paramount.

Guarding development (policies)

ROS development is governed by a number of policies. One example is that development always targets the latest release, with patches being backported to older and *long term stability* (LTS) releases whenever this is considered desirable or required. Another example would be maintaining portability of code in order to remain platform independent as much as possible. Not every newcomer is aware of these policies.

Involve community members in development

As every Open-Source Software project, ROS lives through the enthusiasm of its community members. But this also requires that community members can participate in the development, learn about the design and code structure and can earn merits in order to take on more responsibility in future. Contributing to core packages is an important step in the onboarding of new community members.

Educate community members

To successfully contribute, newcomers need to learn how to write high quality software in-line with ROS' best practices. These principles can best be taught based on actual development rather than only publishing them on the wiki and expecting newcomers to read, understand and apply them before writing and submitting their patch.

Maintaining efficiency

Blocking contributions from being merged into the main code base for too long can be detrimental, both to the engagement of the submitter, as well as to the chances that they will get merged (as a patch may have been written for an older version of the software, increasing the effort required to make it compatible with the current state). As such, Pull Request reviews and iterations should be efficient, with minor (cosmetic) issues not holding up the process.

Solution

Community members as *Submitters* submit a change through a *Maintainer*. The Maintainer should guard the quality of both the contributions and the result of merging the contribution with mainline by making use of their understanding of ROS and of the automated quality assurance tooling.

Stakeholders

A **Maintainer** is either part of the core ROS development team, or a well reputed community member who has taken on the responsibility for a (number of) core packages. The Maintainer 'owns' the respective repository on a ROS Github organisation. His interest and task is to make sure that changes to this repository adhere to the coding standards, that the development guidelines and policies are followed and that introduction of changes by the community does not diverge from the overall design of the software components affected (see also MaintenanceGuide on the ROS wiki).

A second interest is to involve a growing number of community members in the development and to educate newcomers about ROS development guidelines and policies, so that they eventually will be able to not only develop better patches but also might be able to take on more responsibilities.

A **Submitter** is every other community member who contributes for instance a bug fix to a core package. These could be developers using ROS to develop applications and encounter a problem or could also be developers of drivers or maintainers of other core packages. The submitter creates a Pull Request containing the patch and – ideally – a unit and / or regression test for the new or affected functionality.

Tools involved

Github is set up to require approved code reviews of Pull Requests by maintainers before such PRs can be merged into mainline.

The **ROS Buildfarm** runs **Continuous Integration** tests on all PRs submitted against core repositories, and Github is configured on core repositories to require successful test runs before PRs can be merged.

Maintainers can execute components locally to do **runtime or acceptance testing** when appropriate (for instance because an automated integration test is not available or possible due to hardware requirements).

Example

A user of the roscpp client library identifies a bug in queue management (wrong order of operations) that results in messages being lost and submits a PR to remedy this after finding the cause and testing it locally on his own machine[1].

The maintainers of roscpp now need to merge this patch, after making sure it is safe to do so.

Example resolved

The pull request is reviewed, updated and finally merged by two maintainers of roscpp, which are referred to as MA and MB respectively. The chain of events in the review, polishing and final merge of the PR (and related PRs) is as follows:

- user identified problem in a core package (ros_comm/actionlib)
- he diagnoses it, writes a patch and tests that locally
- he then submits a Pull Request against the indigo-devel branch (second-to-last LTS)
- MA identifies some small omissions in the submitted patch and proposes some fixes
- MA creates a new PR based on that of the submitter, as the original PR did not target the correct branch (policy: PRs should target latest ROS release). Submitter's PR is closed in favour of that one. The new PR also includes a regression test contributed by MA.
- MB reviews the replacement PR by MA (policy: many-eyes) and waits for the Continuous Integration server

[1] https://github.com/ros/ros_comm/pull/1054

to complete the tests.

- MB merges the replacement PR into the latest development branch.

At this point maintainer B decides to *backport* the merged fix to other development branches for older ROS releases. He will do that at a later time together with other fixes for which backporting is desirable.

Links

List of core modules

The list of packages which are considered part of the core are documented in REP-142, sections *ROS Core* and *ROS Base*.

Best practices

- For developing ROS libraries and core components see the ROS Developer's Guide.
- For maintaining and releasing ROS libraries and core components see the ROS Maintenance Guide.

Consequences

There must be at least one maintainer per core package (but preferably more).

Maintainers need to have access to the necessary tools, both locally and remote (CI output of ROS Buildfarm).

It becomes possible to add more Q&A tooling to the buildfarm to more easily enforce Q&A process / best practices (make it less subjective).

Known Uses

Many open-source systems have assigned the responsibility for the quality of core modules to so called maintainers, among them the Linux project: https://github.com/torvalds/linux/blob/master/MAINTAINERS

Related patterns

Continuous Integration Testing

Test automation.

Pattern 1: Submit a pull request