Development Process and Quality Control

Introduction

- **Audience:**
  - This document is intended for every developpers that want to contribute.
- **Objectives:**
  - It gives to newcomers the keypoints to start working on OpenERP
  - It defines several rules that have to be followed in order
    - to ensure a good quality code.
    - to ease the community work.

Community Framework

- Open ERP's developments are managed by the Distributed Version Control System Bazaar
- Read the bazaar tutorial before developing on Open ERP. On Windows, you can use the Tortoise BZR GUI.
- Community's developments are hosted on launchpad
  - Create an account on Launchpad
  - Join the community team

Launchpad branches

- OpenERP official branches are
  - **Server**

    lp:openobject-server
  - **Addons**

    lp:openobject-addons
  - Extra-addons

    lp:~openerp-commiter/openobject-addons/trunk-extra-addons
  - Addons from community

    lp:~openerp-community/openobject-addons/trunk-addons-community
  - Check also those projects: **GTK Client**, **Web Client**, **BI**

Get branches

- To download all official branches, do:

```
# bzr branch lp:openerp
# cd openerp
# ./bzr_set.py
```

- You can also get only the branche that interest you, for example the community one, do:

```
# bzr branch lp:~openerp-community/openobject-addons/trunk-addons-community
```

Branch Management

- As our common goal is to capitalize every contributions
  - Don't create branch containing only your modules.
  - Always branch extra or community addons, and add your modules.
- It allows to easily push your modules back to the official branches to automatically
  - test your developments by the integration server
  - update the documentation at http://doc.openerp.com

OpenERP

The first module

- If you're member of the OpenERP community, you can modify or add modules stored in the community branch.
- New features must be developed as modules.
- Create your module and inform bazaar about it:

```
# cd trunk-addons-community

# bzr add <module_name>
```

- Your module will now be commited by your next commit

The first commit

- Once you've done your modifications or add new files, you can commit. Do:

```
bzr ci -m"<commit_message>"
```

- This command commits your changes locally, not on the official branch
- If you want to push your modifications into the official branch, do:

```
bzr push
```

- You should push periodically your developments in order to ease the community work and to benefit of the integration server

Commit messages (1/2)

- Commit messages are used to generate automaticly the changelog at each new version.
- That's why you have to respect the following rules

1. Always set the author's name, if it's different from the committer. Use

   ```
   bzr commit --author="<author_name>"
   ```

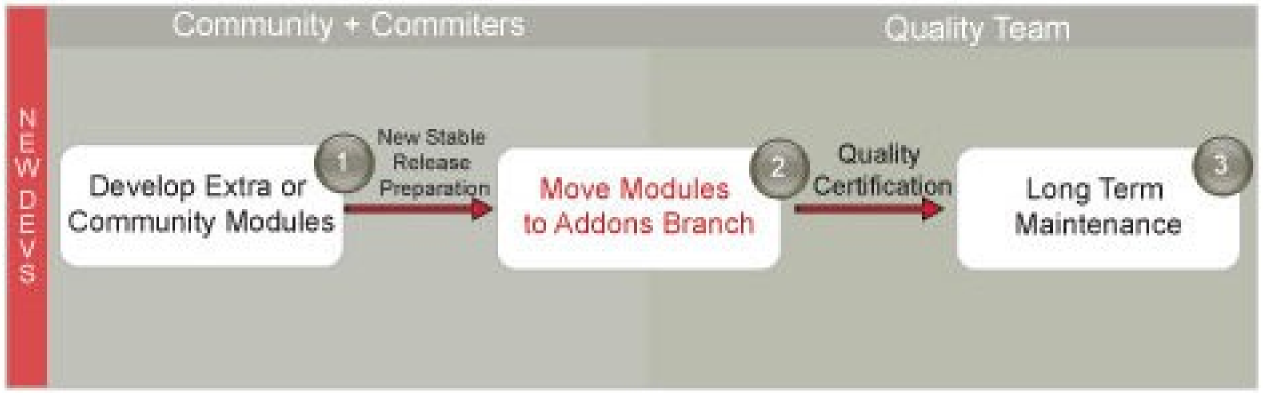
2. Always put meaningfull commit message including the name of the module impacted.
3. If you are fixing a bug encoded in Launchpad, use

   ```
   --fixes=lp:<bug_number>
   ```

Commit messages (2/2)

4. Use header in each commit message.
   - ✓ [IMP] : For improvements
   - ✓ [FIX] : For bug fixes
   - ✓ [REF] : For refactoring
   - ✓ [ADD] : For adding new resources
   - ✓ [REM] : For resources removal

Develop new features (1/2)

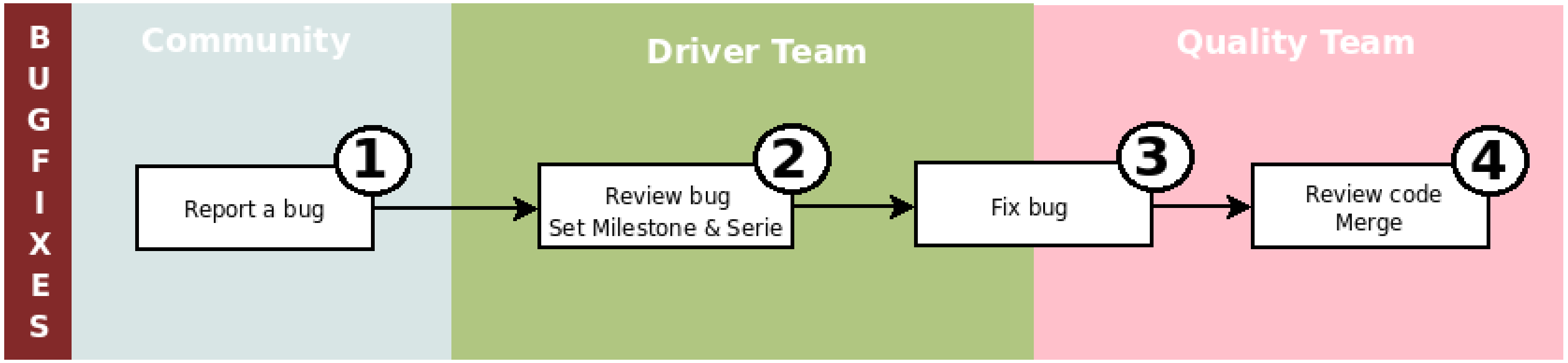

Community + Commiters
Quality Team
NEW DEVS
Develop Extra or Community Modules
1
New Stable Release Preparation
Move Modules to Addons Branch
2
Quality Certification
Long Term Maintenance
3

- New features are developed as modules in extra-addons or community-addons branches.
- Comitters have the right to move the modules from community-addons to extra-addons.
- Before a new release, the quality team will check and certify the best modules from extra-addons to integrate them in the official release.

Develop new features (2/2)

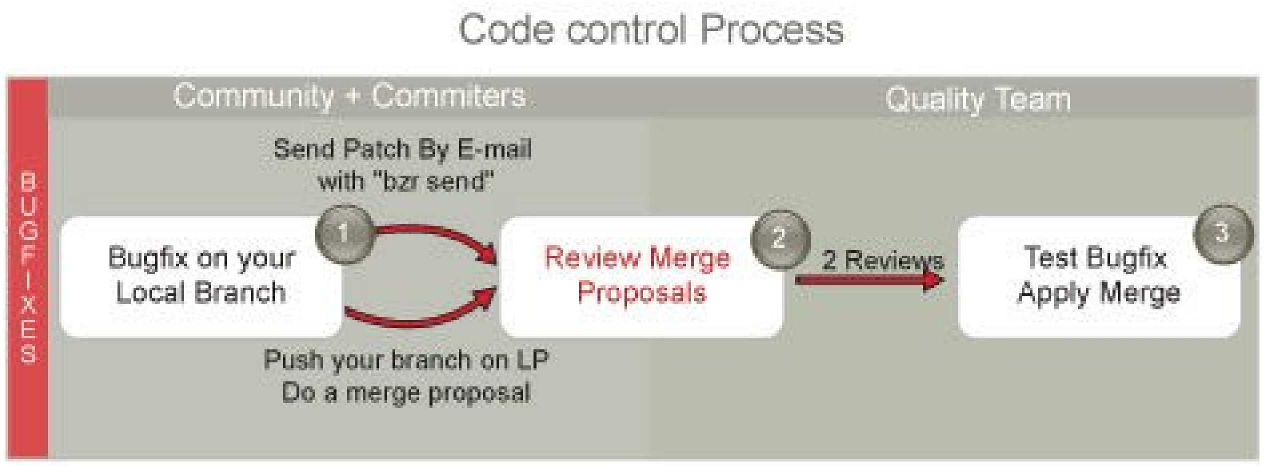
- To develop quality modules, you should be aware of the development guidelines.
- Note also that you can use the base_module_quality module to run automated tests on your modules.
- To guarantee the quality of the official release, and the Odoo version, we have a quality certification process where we check the functional and the technical quality of the module in order to be sure it can be maintained and migrated from versions to versions.

Bugfixing Process (1/2)



1. Someone report a bug
2. OpenERP Drivers review new bugs, confirm and plannify them on a milestone.
3. Developpers fix confirmed bugs assigned on current milestone.
4. If the bugfix must be integrated into an official branch:
   - ✓ Quality Team review the code and merge

Bugfixing Process (2/2)



- **If external people from the quality team want to help improving the kernel, they can**
  - **do their own branch, push it to launchpad and activate "*Propose for merging*" on Launchpad.**
  - **Or they can also directly work in the main branch and simply use the bzr send command.**

Kernel Code Control Process

- **For changes in certified modules:**
  - **Developpers can not merge their changes directly**
  - **Each commit have to go through a code review phase done by someone else than the comitter:**
    - **Either by making a merge proposal**
    - **Or by using '*bzr send*' (merging proposal by email)**
  - **Each merge proposal**
    - **Need to be approved (code review from the diff)**
    - **If approved, be merged (merged locally, test, push)**

API Changes

- Because it's causing problems for every modules that inherit yours, try to never change the API
- If modifications in the API are really needed
  - an announcement have to be made on the OpenERP Code Tracking Blog.
  - This measure allows people to be aware of developments that may cause failures in their modules.

Integration server (1/2)

- Every developments made on official branches are automaticly tested by our integration server
- This include the extra-addons and community branches
  - Periodically push your branches in order to benefit our integration server

Integration server (2/2)

- Coverage of the integration server
  - Stability of the code on databases building
  - Tests of migration scripts
  - Quality benchmaking of modules
  - Unit tests
  - Automatic email sending to comitters

Module Version Numbering

- Defined in the __terp__.py of the module
- A typical version number is 5.1.2
  - The small number « 2 » is increased at each bugfix
  - The medium number « 1 » is increased at each new feature. In this case, the small number is reset to 0
  - The highest number « 5 » can be increased if there were lots of improvements done
- When a module isn't stable (under v1.0.0), you may not have to increase version numbers.