Policy for Software Component Maintenance & Quality

Gunnar Andersson
GENIVI Development Lead

Jeremiah Foster
GENIVI Community Manager

Version 1.0
Scope

• The document outlines code quality requirements and maintenance processes for all* software hosted on GENIVI GitHub account.
• There are different requirements depending on maintenance level
• These are minimum common principles only – each maintainer retains a lot of freedom to define the details (see later slides), and can of course do more.

*Except if GENIVI GitHub hosts a fork which has a clear upstream location defining its own processes.
Background & Needs

- Promote trust in GENIVI code quality and development processes
- Quickly understand which software projects are still maintained
- Document process/policy to back up our interaction with maintainers
- Agree on *expectations* between GENIVI and code maintainers for *all* projects hosted on GENIVI’s GitHub account
- Accountability to a mutually agreed standard & process for recognizing and addressing any project issues with maintainer (manager).
- *Provide better information* to our community, what to *expect*, and how to resolve issues
Separate concerns

1. The intended level of maintenance on each project (commitment)
2. The current level of delivery on that commitment (evaluation)
3. Communication about mitigation, if there is a discrepancy between 1. and 2.
4. Inform contributors and recruit them to vacant maintainer positions.

All of these aspects should be communicated to our community, although escalating problems might be done first in private with respective maintainers and companies.
Action Plan

1. Document code quality improvement activities
2. Document project maintenance level
3. Quantify expectations (response time etc.)
4. Inform all maintainers about new policy, get feedback
5. Reach out to all documented maintainers - “are you still there?”
6. Inform the community about the policy & process
7. Follow up & support maintainers in implementing the maintenance actions (until it’s flows naturally)
8. Follow up implementation of Code Quality activities
9. Finalize and publish Continuous Integration Plan (related)
SOFTWARE MAINTENANCE
Maintenance - expectation vs. delivery

1. Addressed with two independent but related aspects:

2. First, agree on project expectations / maintainer commitment
3. Then, measure health and highlight problems (delivery on commitment)

- **A) “Maintenance Level”** – Plan & commitment
- **B) “Maintenance Status”** – Project health from maintenance point of view (performance evaluation)

Expectations will depend on agreed maintenance level, but once that is clear, the community expects the maintainer to meet the expectations.

- There could be many reasons for that being challenging, but we should take note of any issues and find a solution together.
## Component Maintenance Level

1. **Maintenance level** | **Expectations**

| ACTIVE         | Response time: 1-3 days  
|                | Resolution time: ~1 week |
| PARTIAL        | Response time: 1-2 weeks  
|                | Resolution time: ~2-4 weeks |
| NOT ACTIVE     | Response time: N/A       
|                | Resolution time: N/A     |

**Response time:** Reading and submitting a first on-topic response to Tickets/Issues, Mailing List questions, and submitted patches / pull-requests.

**Resolution time:** Discussing towards, and reaching, a conclusion to close a ticket, merge (or reject) a patch or pull-request. (Understandably, resolution time can differ depending on the situation).
Less active projects

- Open-source projects progress simply at the pace of the contributing community, which in turn depends on the needs and policies of involved organizations. This is true for all projects.
- Community interest and available resources may come and go over time.
- It might be said that projects that are not actively developed may be of little value to keep. However, reduced activity can depend on many things:
  - The project is temporarily transferring from one maintaining organization to another
  - It is mature and stable, and therefore needs less activity, beyond critical bugfixes – but those appear only if they are noticed, reported and fixed.
  - It may have hidden value but not yet noticed, or the timing is right for system integration
- In GENIVI we believe good source-code has inherent value. Some open-source code may be published despite being NOT ACTIVE, for anyone to use when they so desire.
- All projects are still evaluated for appropriateness and applicability to the GENIVI Alliance scope
- NOT ACTIVE is not a blocker. Anyone interested in a NOT ACTIVE project may be given the opportunity to become a committer/maintainer, and thus be able to progress the code base.
# Maintenance Status & Selection

1. **Maintenance Status** is distinct from communicated **Maintenance Level** – it is about the current maintainer performance. It is a guide for conversation with maintainers. It is also informative to the community, but any problems should be brought privately to the maintainer first.

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GREEN</td>
<td>All OK</td>
</tr>
<tr>
<td>YELLOW</td>
<td>Resolution required: Issues with activity, long response time or other</td>
</tr>
<tr>
<td>RED</td>
<td>Highlighted problems not addressed. Inconsistency with GENIVI goals.</td>
</tr>
<tr>
<td>VACANT</td>
<td>Interim process in place in absence of active maintainer</td>
</tr>
</tbody>
</table>

CODE QUALITY
Software Component Standard – Background

- GENIVI writes, publishes, adopts and reuses open-source software. Some components are adopted or contributed to GENIVI to be stored under the GENIVI GitHub account (a.k.a. “GENIVI components”), but the software platform is also built from many other Open-Source projects.

- Because of the diversity of sources and different levels of connection, applying one single code standard for all “GENIVI projects” is neither feasible or desired.

- Individual projects thrive when driven in the way that the contributors want.

- Maintainers thrive when using their own preferred tools and methods.

- Contributing companies are often obliged to follow their own coding standards.

- In this policy we therefore set common requirements, but leave freedom to the maintainers for a lot of the details.
Maintainer Freedom – Shared Quality

• The GENIVI code projects policy leaves a lot of **freedom for the maintainer** to define the details of *code standard, layout and look-and-feel, git repository organization, working processes, and preferred tools*.

• The common policy however prescribes a few **common characteristics** and **quality requirements**, and it requires *maintainer choices* to be consistently documented.
Component Quality – Continuous Build

1. All components shall have a continuous build system set up to:
   1. Test-build proposed changes (*pull-requests*) before merge
   2. Build *Master/Develop* branch on every change
   3. Publicly accessible dashboard showing build status – (“Build badge” on README.md recommended)

2. For projects containing unit and component tests – continuous running of those tests for changes on Master/Develop branch (with Status badge)

3. *For more details, refer to the separate CI-strategy document* (pending) – the same is preferable, but cannot be required
Component Quality – Static Analysis

1. Depending on tool availability (programming language) perform continuous static-analysis scanning on Master/Develop branch, with status badge.

- **C/C++/Java/similar**: (Required) professional tool, e.g. Coverity
- **Python**: (Recommended) style checker, like PEP8
- **Bourne Shell scripts**: Linter not required, but recommended. bash-only scripts OK, at maintainer’s discretion
- **Others**: (Recommended) similar appropriate tooling

---

**ACTIVE**

**PARTIAL**

---

**NOT ACTIVE** – the same is preferable, but cannot be required
Component Quality – Test Code

1. Every project need is different, but programmed **unit tests** (and potentially component-tests) are *strongly recommended* on non-trivial programs.

2. We should from time to time review all projects and encourage those that could have more unit tests to create them.

3. Unit tests are no different from any other code – all code in an open source project depends on the input of the community that uses (and writes) the code.

4. If the program includes unit or component tests they shall be run – see CI requirements

5. If the component is part of Baseline (**meta-ivi**), tests shall be integrated into the **meta-ivi-test** layer

6. If the program includes unit tests, we **recommend** a code-coverage checker (Coveralls, CodeCov, etc.) and corresponding README badge.

---

- the same is preferable, but cannot be required
Component Quality – Style

- Consistent code style (*maintainer defined*)
- Required: Fully consistent indentation/whitespace style
  - Documented!
  - Do not mix spaces and Tabs
  - Unix line-feeds on all code & data files, unless the program or file is explicitly targeting Windows (e.g. a development tool)
- All other code style details are defined by the maintainer, but it *shall* be defined & documented.
- Recommended: Automated code indenter/beautifier
Component Quality – Documentation

- README.md shall exist – please use this as a template: https://github.com/GENIVI/admin_and_reporting.

  Documented contribution process, issue-tracker, support channels, code standard and policies. Preferably held in a CONTRIBUTING.md file

- Links to user & developer documentation, as appropriate

- Unambiguous license information. (COPYING/LICENSE text file in proj. root & © Copyright & license name in headers)

- Auto-updated generated documentation if kept in source format (e.g. Markdown, XML, etc.) – Suggestion: Use community and GitHub integrated tools (GitHub Pages/Wiki, readthedocs.io, etc.)

- Recommended: Code review to be documented in GitHub tool, alternatively on mailing list.

- Refer to: Community Manager documentation on Public Wiki
END