Metrics Definition Inside the Software Quality Assurance Process

Andres Abad Rodriguez
CERN

CERN School of Computing 2010
Uxbridge, 30 August 2010

Contents

- Why do we need to collect metrics?
- Different approaches
- ISO 25000 standard
- Indicators
- Types

Why do we need to collect metrics?

- Because they are the basic indicators that provide us the necessary feedback to know if our project is going in the right direction.
- They also provide a better understand, track, control and predict software projects, processes and products.

Different approaches

We can select one of the two next approaches (as described in W

- Measure eve measureable, report on it and try t information.
- Collect what e the currently most important me

[1]Linda Westfall, "12 steps to useful software metrics", http://www.westfallteam.com/Papers/12_steps_paper.pdf

ISO 25000 standard

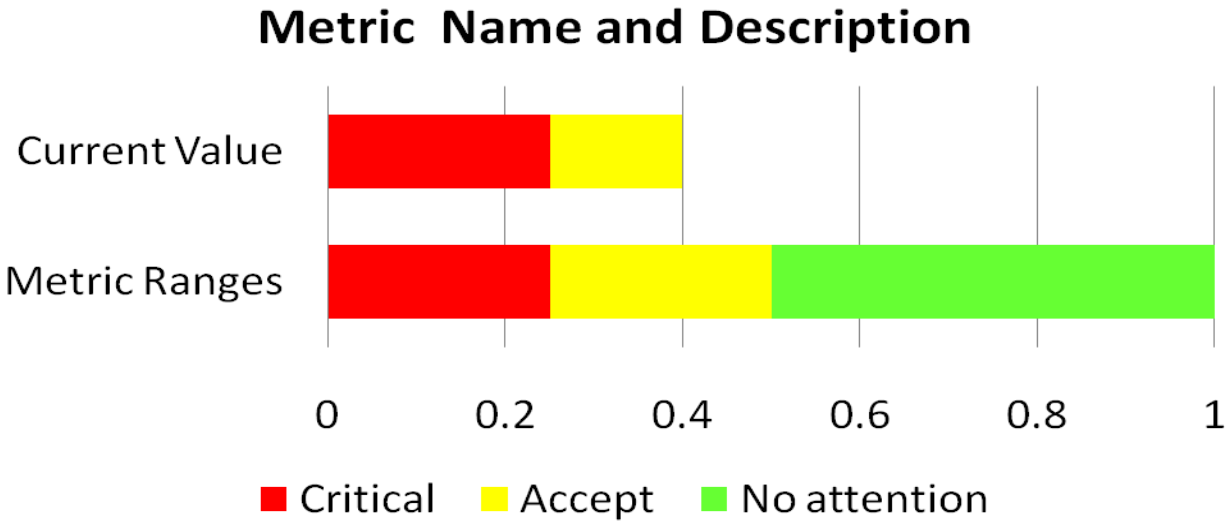
There is an international standard that will help us to define our metrics inside the software QA process: **ISO 25000**.

- Also known as Software Quality Requirements and Evaluation (SQuaRE).
- It assumes that the software architecture design, or software development plan is already in place
- It includes the previously existing ISO 9126 standard

Indicators

The metrics can be plotted as a horizontal bar-chart, typically using a normalized scale [0,1] or [0%,100%] which is split into:

- **Red**: Critical, needs improvement or immediate attention.
- **Yellow**: reasonable or needing some attention.
- **Green**: Very good and needing no immediate attention.



Types

- Last step to define metrics is to select which one of them are important in the project.
- We can divide them in groups: process metrics, product metrics and quality in use metrics.
- Additionally we can add an extra group called “Additional metrics to consider” to add some other metrics necessary for the project but can not be placed in any of the other groups.

Types (II): Process metrics

- Average time to handle a high severity bug
- Bug severity distribution (grouped by severity level)
- % of failed nightly builds
- Certification test effectiveness (% of bug found during integration test of a release over the number of bugs found in production of the same release)
- Up-to-date documentation
- Delay on the release schedule

Types (III): Product metrics

- Unit test coverage
- Memory leak warnings
- Number of supported platforms (from the total defined to be supported)
- Total bug density (number of bugs per KLOC*)
- Total bug density per release

*** KLOC: 1000 lines of code**

Types (IV): Quality in use metrics

- Total user incidents* per period of time
- Training and support incident per period of time
- Average time to deal with an incident

•**Incidents are all user communications that are not software bugs**

Questions