Maintenance Effectiveness Indicator

Document update: Oct. 27, 2023

It is not easy to answer a simple question:

Are we spending enough (or even too much) effort on maintenance?

Estimating Maintenance Efforts Up-Front?

Estimating how much maintenance work will be needed for a component up-front is a hard task. Consider that there are different types of work that can all contribute to improvements in our code-base, like:

- Direct bug-fixing of reported errors
- Architectural improvements
- Refactorings (for example pointer usage improvements)
- Code removal (old, unused features)

It is not easy to say which of these measures will contribute most to our overall maintenance quality (and the answer might be different from component to component, too).
It is also not easy to estimate their effort, in particular we cannot know up-front how many error reports we will get (and how serious/complex they will be).
But at least we can hope to have enough information from the past that can tell us in general, if we are doing enough here.

Zoom out: What is “Maintenance Quality” here?

We used the term “maintenance quality” above, but in order to avoid confusion we should be clear about what aspects we want to include in our “maintenance quality” here and what not. We can summarize this probably as:

Maintenance quality is the level of reliability for the user with which we deliver a feature within a software artifact.[1]

[1] Think of “feature” as “something we promise to be working”. Obviously it is not always easy to point at the concrete and complete definition of this, but that doesn't really matter for this indicator to be meaningful. Still during bug triage we (some domain expert) usually know if this bug is a defect or not.

Please note that this way of defining quality excludes some aspects, that users would normally include in general “Quality”, like:

- Fitness for purpose
- Completeness of supported standards/functions
- Design, UX
- Formal quality of our processes (like ISO standards)

Instead it includes aspects like:

- Defects
- Crashes
- Security/privacy problems

In order to remind us of these limitations, we thus want to talk about “maintenance quality”, not just “quality” in general.

What is “Maintenance Effectiveness” then?

We currently have no way to directly measure “maintenance quality” based on our data and processes in a meaningful and absolute way. A relative measure is “maintenance effectiveness”. It assumes that we can measure if we do enough in order to keep the maintenance quality of our software artifact at the same level (or even improve it).

How can we measure “Maintenance Effectiveness”?

Let’s see which measurements we have at hand in order to judge our maintenance effectiveness.

We produce a huge amount of data we could look at:

- Crash Stats
- Mozilla Bugzilla
- Treeherder
- Various bots (intermittents, fuzzing)
- Other failure dedicated telemetry (like QM_TRY)

But at the end of the day, all different ways of recognizing a failure will result in a bug in bugzilla. And this bug will be most likely labeled as a **defect** (see What is actually a “Defect” ? for caveats).

Now bugs can have all kinds of complexity, severity and such. **But in a nutshell a newly opened defect bug is always claiming that “there is something more that does not work as expected”.**

However, the absolute number of open bugs on a component does not tell us much about the component’s perceived quality. It might well be a huge mountain of technical debt, but it might also just be a badly maintained backlog. And in the end we are not particularly interested in absolute measures, anyway. But we certainly want to know:

Are we losing or gaining Effectiveness ?

We want to look at the trend. Assuming that all our processes work as expected (triaging, automatic testing, …), we can certainly see a trend over time in the data even if we look at the most trivial indicator we can imagine:

If $OpenedDefects(\Delta Time) > 0$:

$$ME(\Delta Time) = \frac{ClosedDefects(\Delta Time) * 100\%}{OpenedDefects(\Delta Time)}$$

Else

$$ME(\Delta Time) = (ClosedDefects(\Delta Time) + 1) * 100\%$$

Note: The +1 just ensures that for the case of 0 closed defects we get 100% and anything above 0 is (very) positive. But generally speaking the result is of low value if there are no incoming defects at all.

A value > 100% means we are gaining effectiveness (and thus hopefully are improving the quality of our software artifact). A value < 100% means we are losing quality.

ΔTime is important

We are looking at a trend here. So we do not just want to look at the most recent ME(ΔTime) for let's say the last week. We want to see how we move the needle over time here. A way of doing so might be to have side by side:

ME(last week)	ME(last month)	ME(last 3 months)	Trend
120%	110%	80%	We turned the trend, hopefully?
90%	110%	100%	Just noise
80%	90%	120%	Attention, we might start to lose quality!

Weights

We weigh bugs based on their severity[2] as follows:

Severity	Weight (Fibonacci)	Rational
S1	8	We rarely expect to see these in our backlog, but if they appear, they outweigh anything.
S2	5	This gives continuity to the S2 burndown efforts of 2023.
untriaged	3	Triage is important, but less important than S2 in terms of "effect on quality".
S3	2	The vast majority of our defects.
S4	1	Most intermittent failures are here, for example.

Which changes our formula to:

If $WeightedOpenedDefects(\Delta Time) > 0$:

$$ME\%(\Delta Time) = \frac{WeightedClosedDefects(\Delta Time) * 100\%}{WeightedOpenedDefects(\Delta Time)}$$

Else

$$ME\%(\Delta Time) = (WeightedClosedDefects(\Delta Time) + 1) * 100\%$$

With

[2] This is assuming that our bug's severity is well maintained over its lifecycle. See for example https://github.com/mozilla/relman-auto-nag/issues/1304.

$$WeightedClosedDefects(\Delta Time) = \sum_{i \in Closed(\Delta Time)} Weight(Severity_i)$$

$$WeightedOpenedDefects(\Delta Time) = \sum_{i \in Opened(\Delta Time)} Weight(Severity_i)$$

What is a good target value for ME?

In general, 100% is probably a good target that tells us “we are on top of all incoming problems”. There might be situations where we decide that a given component is deprecated and thus we do not want to spend any effort on maintenance or on the contrary we want to deliberately increase the quality of a component and thus we want to set ourselves a higher goal.

Estimate the time to burn down all the defects backlog

If $ME(\Delta Time) > 100\%$ we can **estimate** the time needed to burndown the defects backlog:

$$eBDTime(\Delta Time) = \frac{WeightedOpenDefects(Now) * Duration(\Delta Time)}{WeightedClosedDefects(\Delta Time) - WeightedOpenedDefects(\Delta Time)}$$

Otherwise the BurndownTime is ∞.

Note that we can calculate this value also on the unweighted defect numbers, and we will do so in order to be able to confront both. But it is expected that for the time being the weighted variant gives better comparability to ME, though we want to assess the accuracy of the estimations of both. The weighted variant kind of assumes that it is harder and requires more effort to close a bug with high severity, which may not always be true (a fuzzer found UAF is severe but might be very easy to fix), but on the other hand it is probably more likely that lower severity bugs will be closed without any effort (WONTFIX or such), too. We have no data on spent effort on single bugs that could help to assess this further.

So what do we actually measure here?

We are measuring the **effectiveness** of our maintenance **process** from triage to fix or otherwise closing of defects in relation to incoming bugs. This has some implications:

- A component that never receives new defects will always look good, regardless of the existing (old) technical debt.
- A component that has been newly added might receive an initial uptick of incoming defects as part of the creation process
- Backlog cleanup (old bugs get closed) can cause positive spikes
- Start of a new defect finding tool (like fuzzing, code checker) can cause negative spikes

This is thus a very indirect measure of the resulting maintenance quality of our product, based on the assumption that **a newly opened defect bug is always claiming that "there is something more that does not work as expected"** and that all kinds of problems we know and care about sooner or later will become a defect in our backlog.

What can we do with this Indicator?

Motivate people to do backlog cleanups

This indicator is pretty sensible to backlog cleanups (see One time effects). That makes it a good tool to motivate people to do backlog cleanups until we reach a backlog state that makes it hard to have those "low hanging fruits" of old bugs that can be easily closed. Once we reach that state, we can:

Steer against the trend

We can use this indicator in order to steer against the trend. If we constantly see numbers < 100% we should probably do something about it.

Combine the past Trend with past Effort for better Estimations

Remember the Estimating Maintenance Up-Front? Paragraph which was our starting question. Most developers have a quite good feeling for how much time they spent (in % of their time) on maintenance. Please note that this might also include tasks that are explicitly not directly included inside this indicator's calculations, like refactorings, and that this is wanted. We can then use this percentage to calculate a single developer's effort for a given time in the past and sum this up for a team.[3]

[3] Please be aware that to sum up these % values to real efforts (person months) in practice is a non-trivial task. See DOM LWS Backlog vs. People H1 2022 second sheet "People H1 2022" as an example,

Once we are able to estimate the maintenance efforts the team spent for a given, past time period, we can estimate how much effort we should have been spending:

$$NeededEffort(\Delta PastTime) = \frac{SpentEffort(\Delta PastTime)}{ME(\Delta PastTime)}$$

We can then project this value into the future for our planning. Please note that this gives us an absolute estimate of effort (under the assumption that the set of components does not change) that will be most accurate if executed by the same set of persons that was doing the work in the past.

Know if we should check the Quality of our Efforts

If we are convinced to have put enough effort into maintenance but the trend is still bad, we probably did not do the right kind of maintenance. The indicator cannot identify the error, but it can help to raise attention if we should take a closer look. We could for example compare the situation with teams/components that do better in order to understand the difference.

Fuzziness and Caveats

What is actually a “Defect” ?

Bugzilla contains many bugs that are marked as a defect but will probably not hold against the limited concept of being a defect in the sense of “maintenance quality”, thus **affecting the reliability during the usage of a feature that is part of our value proposition**. For example many users would deem the missing support of a feature of a web standard to be a defect, not an enhancement, so they would file the bug as defect.
It should be part of the triage process to keep an eye on this question (“is this really a defect?”). But in the end it is not expected to be very relevant: The number of new bugs that are asking for new features in web-platform is probably not high enough to matter. If we have some of those bugs in our data from time to time, they will most likely not have much influence on the trend if we look at it frequently. And if we happen to close some of them eventually, the indicator will be better, too. So this fuzziness should level out over time, at least for long-time existing components (and on a completely new component this indicator is not meaningful anyway - there is just nothing to maintain, yet).

When can we consider a bug to be opened?

If a bug has been created within ΔTime.

When can we consider a bug to be closed?

If a bug is currently (at the time of the query) RESOLVED/VERIFIED/CLOSED and the resolution has been set within ΔTime. There might be edge cases where the resolution is set more than once during time, but these should level out after a while (if we re-issue the query on life bugzilla data anytime we need it, of course).

For now we do not consider tracking/version flags here. See also Should we have weights for release channels?

Does it matter how old a bug is when we close it?

No.

Is the way we close a bug important?

Bugs can be closed in many ways:

- FIXED with a patch
- As INVALID or WORKSFORME
- By some bot (intermittent failures and such)

While the amount of work we had to put into a bug in order to get it closed can vary from zero to almost infinite, it is still closed. Even random INVALID bugs contributed to a negative quality perception as long as they were open. The earlier we filter out the noise during triage the better - not only for our statistics. Intermittents will eventually go away due to other changes (be they targeted or not). All this counts.
However we could lower the severity automatically to S4 if we close a bug as INVALID, INACTIVE, INCOMPLETE, EXPIRED, DUPLICATE (in order to reduce their weight). But as long as they are open they claim there is a defect.

How do other changes on the bug affect this?

Bugs can change in many ways, like changing the component, severity, re-open it and so forth. It is thus important to run the queries on the most recent bugzilla data again even if we want to compare historical data from different time intervals.
(The case that the resolution is set more than once is probably an edge case that would not be treated well. On the other hand, we would just consider the latest change here as a resolution time stamp, which is probably ok in the sense of “we really understood only now what this was”.)

One time effects

There might be situations where we do things that heavily affect ME(ΔTime), like:

- Backlog cleanup (old bugs get closed)

- Start of a new fuzzing tool (many new defects)

These might result in spikes for some ΔTime. Generally speaking these events correct wrong assumptions about our quality (before the backlog cleanup we would look with more doubt at our component's health, while the new fuzzing tool uncovered unknown problems we were just not aware of) and thus should not be excluded from counting. But they will level out after a while and are expected to be rare enough for single components to not really matter.
The assumption/hope here is, that once we introduce this indicator, people will start to pay more attention to their backlogs and will apply low hanging fruits asap. Once they reach a more stable situation and only the "hard work" is left, the indicator will automatically stabilize, too.

Should we count (refactoring) tasks? - No.

Besides "defect" and "enhancement" we also have the category "task" in bugzilla. It is mostly used by developers to designate some work that needs to be done but that usually has no direct impact on the users. It is also used to mark some automatically generated bugs (for example [wpt-sync] bugs).
In some cases these tasks are refactorings that improve the reliability of the code significantly. Still they have no intrinsic value in the sense of our indicator: if there is no visible defect there is no problem to solve. But they might well contribute to having much less new defects (intermittents, crashes) than in the past, though.
So the short answer is: no, we shouldn't count them.

Should we have weights for crash/intermittent frequency? - No.

We can hopefully expect that a high frequency crash/intermittent will get a high severity rating in bugzilla. Furthermore crash numbers change over the lifetime of a bug in a significant manner such that it would not be easy to understand when to sample this data. So we cover this with the weights on severity (see above).

Should we have weights for release channels? - Not for now.

When bugs get fixed, the fix reaches users of different channels at different moments. We track this by release flags. We could want to use this information in order to have weights. This would help to avoid problems in saying "Can we consider this bug already closed?". Note that this weight should apply only for the ClosedDefects(ΔTime) calculation, opened defects would always count as 1 here.

TODO: Think of a meaningful way to calculate these weights. Starting from something like:

- No fixed flags - full weight
- Fixed in future release - ⅔
- Fixed in past release but future ESR - ⅓
- Fixed everywhere (or otherwise closed) - 0

Besides the weights the question is how to deal with other values of those flags, like “disabled”. In general this is probably quite complicated given the current way of managing those flags in bugzilla.

What about P5/S4 or otherwise unimportant stuff?

Part of the answer (S4) has been given through weights. For the rest: If we just decide to not work on something (P5 or so) this does not mean it is not affecting quality. In fact, if we decide very often to do nothing, we will decrease the quality constantly. And that is what we want to measure here.

Should we look at different aggregations of bugs through this lens?

In theory this indicator can be used for any meaningful grouping of bugs that is stable over time, provided that it is large enough to not just see noise and sharp enough to be actionable for measures. In practice it is probably expected to refer to some well identifiable software artifact whose quality we want to monitor in order to be meaningful. Some possible aggregations:

Security bugs - No

We probably already have enough insights here. And calculating this index over all components is not very actionable. The absolute numbers are too low to calculate it over single components (remember that we talk only about opened vs. closed bugs). In any case **security bugs do not map meaningful to software artifacts**.

Regressions - No

While it is important to know if something is a concrete regression of another fix in order to determine the cause, this distinction adds little value to our indicator: What would I learn if I see that we fix well regressions but have a low effectiveness index in general? Still I spend not enough time on maintenance. This indicator will not be able to tell me what exactly I should do more, anyway. And again also **regressions do not map meaningful to software artifacts**.

Components - Yes

This is probably the most stable anchor we have in bugzilla. **They usually refer to some kind of software artifact** and are the entity we define triage ownership upon which helps to make the results actionable, too. We also might want to set special goals (!= 100%) for our indicator on single components (see “What is a good target value?).

Teams and higher organizational Levels - Yes

The same aggregations on components as in bugbug would apply well here and keep things actionable.