Table of Contents

1  A common trap in software quality management projects: deploying too many abstract indicators ..................................3
1.1 How to lead a Software Quality Management project ........................................................................................................3
1.2 The trap of deploying too many abstract indicators ...........................................................................................................3
2  How to define efficient quality indicators ...........................................................................................................................................3
2.1 What is clearly thought out is clearly expressed ..........................................................................................................................3
2.2 Characteristics of an efficient quality indicator ...........................................................................................................................3
3  Technical Debt-based indicators: relevant, ready to use and easy to deploy ..................................................................................4
3.1 What is Technical Debt ...................................................................................................................................................................4
3.2 The advantages of Technical Debt indicators ..............................................................................................................................5
4  The Squore indicators to manage your Technical Debt .....................................................................................................................5
1 A common trap in software quality management projects: deploying too many abstract indicators

1.1 How to lead a Software Quality Management project

When talking about Software Quality, most of articles cover best practices and development process to implement, such as identify risky components, do frequent code reviews, refactor complex artifacts, check coding rules, etc.

However, few guidelines can be found around the project itself. Yet, deploying a solution for Software Quality Management should be considered as a project, with its stakeholders (project manager, developers, QA manager, subcontractors), milestones and - like any other management project, the common traps to avoid.

Thus, like in any other management project, indicators are strategic and will reflect your quality approach by making it concrete. Indicators are the main expected deliverables, and this is especially true for a Governance Program of your software developments.

1.2 The trap of deploying too many abstract indicators

In this context, the « trap » lies in the temptation to express the quality of a software artefact with a magical synthetic indicator, which would be built from numerous heterogeneous and conflicting data. In short, badly designed indicators would be like hieroglyphics in your dashboards: no one can understand them, even their own designer!

And when we fall into the trap, we usually get an unbounded value with no measure unit. In example: my application overall quality equals to 128, 72 for maintainability, 42 for robustness, etc. Doesn’t it ring a bell? As you’ll easily understand, these values do not mean much, taken out of their context. Moreover, they quickly raise some embarrassing questions:

> How should we read the indicator? Is the quality improving when the value increases or decrease?
> How more serious is the situation when the value is 132 instead of 128?
> Is the robustness value comparable with other projects?
> What real effort is required to improve by 10 points the maintainability value?

And there are many other questions that will quickly highlight the poor design of your indicators and will weaken your quality management program at an early stage:

> Problems for developers and managers to share an understanding and communicate with each other
> If an indicator is misunderstood, it won’t be adopted. It might even be fully rejected
> The tool computing these indicators might be challenged, replaced by « homemade » indicators: this double reading will jeopardize your decision making and the resulting action plans.

2 How to define efficient quality indicators

2.1 What is clearly thought out is clearly expressed

The design of your quality indicators is a key step that needs time and attention. It must be clear, strong and indisputable, in order to allow your quality program to be understood, applied, adopted and continuously improved. Remind its essential role before building it: an indicator measures the gap between a situation and a target to be achieved, and helps you lead actions to reach your quality goal.

For example, one of my primary goals for my mobile application is its reliability during the exploitation phase:

> I will make sure that the code is easily testable
> I will also check if it has been effectively tested!
> My reliability indicator will be based on the following data:
  > Code testability metrics (methods cyclomatic complexity, coupling between classes, etc.)
  > Unit Test information, such as code coverage
  > Data from function testing campaigns
  > To complete the picture of the reliability level of my software, I could also include information from load testing tools. Indeed, if my app is not able to support thousands of concurrent users, unavailability due to performance and scalability issues would be considered as a lack of reliability.

2.2 Characteristics of an efficient quality indicator

Without diving into the specifications detailed in “NF X50 -171” or “ISO 9001:2008”, we could say that an indicator is efficient when it meets the following requirements:
Relevance and usefulness: How well does the indicator represent the gap between the situation and my quality target?

Simplicity: A good indicator should be explained with a few words. Say what it does, do what it says, then check. It also must indicate if the target has been reached or not (see the « Goal Question Metric » approach by Victor Basili).

Representativity

Comprehensive: The indicators must be available for each level of your artifacts (portfolio, app, package, etc.). Actually, this point is made easier to cover with static code analysis tools that automate data collection.

Quantifiable: For example, code coverage metrics, total bugs found in execution, rule violations found in code, sum of new code lines delivered, average of days of delay for a delivery, etc.

Objective: The components of your indicators shouldn’t be controversial. What about the number lines of code to measure the size of a software artifact: Should we count blank lines? Commented lines? Only compiled instructions? Generated code should count as part of the software?

Deployability: It’s one thing to sketch your indicators on paper, but quite another to deploy and to make them « live » with real data. Often, the data is not in a proper format, scattered from different tools, or even does not exist at all for some artifacts. You will have to ensure availability and scalability of your indicators.

Basically, an indicator is alive: it is created, it evolves but it can also die if you think it is ambiguous or inefficient in production. Don’t hesitate to kill your weakest indicators, according to the characteristics detailed above; it is like a natural selection process. The sooner the better, because it is much harder to retire an indicator which has been deployed for your whole project portfolio.

3 Technical Debt-based indicators: relevant, ready to use and easy to deploy

3.1 What is Technical Debt

Technical Debt indicators are particularly appropriate to start your Software Quality Management project. Initially invented by Ward Cunningham in the 90’s, the Technical Debt comes from all these tasks (feature, bug fixing, code refactoring, architecture optimizing, etc.) that a development team would -whether willingly or not- delay to a further sprint or product release. Delaying tasks immediately provides the team with a breath of fresh air, but like monetary credit, the later you’ll reimburse the higher the interest rate will be. In other words, if a feature costs “100” to be developed today, it will cost “100 + something” tomorrow, “100 + something higher” the week after and so on.

Philippe Kruchten (Professor at the University of British Columbia, Canada) recently enlightened the metaphor with the matrix below, positioning the Technical Debt as a negative and invisible element that depreciates the value of a software:

<table>
<thead>
<tr>
<th>Positive value</th>
<th>Invisible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Features</td>
<td>Architecture</td>
</tr>
<tr>
<td>Negative value</td>
<td>Bug</td>
</tr>
<tr>
<td>Technical Debt</td>
<td></td>
</tr>
</tbody>
</table>

© Philippe Kruchten

Many IT companies (such as Vector and Inspearit with the SQALE method) have implemented this concept to provide development teams with tools to identify and quantify the amount of the technical debt of their applications, generally expressed in days. Thanks to these measurements, this kind of indicators become powerful tools to pilot the quality of software development projects.
3.2 The advantages of Technical Debt indicators

- Usefulness: Technical Debt heavily weighs on team agility and innovation capacity. A high technical debt means important efforts allocated to corrective maintenance, to the detriment of the added value of new features delivery. A tech debt indicator will highlight the quality gap of software artifacts, with respect to product quality requirements (robustness, security, reliability, maintainability, etc.) and nonconformities found in the project.

- Ease of Understanding: Technical Debt speaks for itself when displayed in UOW (time, money…), from the CIO to the developers through DevOps. Workload of each postponed task -or delivered but with an unexpected level of quality- is summed up to measure the total debt of an application, a project or an artifact. For example: we’ll need to allocate 10 man-days to make the project conform to reliability requirement, as defined in our quality model.

- Ease of Deployment: At its simplest, technical debt is computed by cumulating workloads of nonconformities. Thus, your first attempt to measure projects debt can be done into just an Excel sheet.

- Reconciliation of technical and managerial visions: By providing the stakeholders with a relevant, understandable and non-ambiguous information, they use a common framework to act on real software quality issues, rather than discuss the definition of the indicator itself.

- Task-oriented and objectified, by nature: this is a huge advantage of the Technical Debt indicators. Computed from non-conformities found into the project (code, documentation, requirements, etc.), it allows you to list the necessary actions to reduce your technical debt with an extreme ease, in an efficient and objectified way.

4 The Squore indicators to manage your Technical Debt

Squore/Software Analytics includes packaged Technical Debt indicators built on extensive experience in Software Quality Management projects. In minutes, the solution allows to deploy a primary level for an effective management of Software Quality.

Being expressed with remediation units, the technical debt computed by Squore easily indicates the level of quality risk of analyzed artifacts.

Squore’s indicators enable an easy and objective way to monitor your applications’ quality, whether they are in a development or maintenance phase:

- An easy to understand rating system (color-coded A, B, C, D, E and F grades) providing accurate information (technical debt estimates in man-days or currency…)

- Fast highlighting of risks and quality issues in a project portfolio, thanks to unique drill-down features through the quality characteristics and into the software components (projects, packages, methods, functions, etc.)

- Indicators taking into account additional types of data (e.g.: functional requirements, change requests, etc.).

- Automated Action Plan generation, optimized to efficiently reduce Technical Debt of your projects, based on standard and project-custom quality requirements.
Get More Information

Visit our website for:
> News
> Products
> Demo software
> Support
> Training classes
> Addresses

www.vector.com