Managing Technical Debt with the SQALE Method

by Jean-Louis Letouzey

Since its publication in 2010, SQALE[1] has become the industry standard method for managing technical debt. This open source, royalty-free method is implemented by multiple static analysis tools, including the SonarQube platform,[2] which is used in more than 50,000 companies, with an estimated 2 million users.[3]

This article will present the key concepts of the SQALE method and explain how to use it, either in a day-to-day context (as, for example, within an Agile project) or at corporate level to govern a portfolio and optimize its technical debt. The main goals of the method are to:

- Provide a rough estimation of the principal and interest of the technical debt of a piece of source code. It could be a small piece like a file or a complete IT domain build of numerous applications.
- Provide indicators that allow detailed analysis of the nature of the technical debt.
- Support remediation strategies with relevant indicators. As we will see later, there is no one magic strategy for paying back technical debt. There are many potential strategies, and the right choice is highly dependent on the context.
- Be implementable within an automated solution in order to provide real-time visibility and decision support.

To achieve these goals, the SQALE method uses four concepts:

1. A quality model
2. Estimation models
3. Indices
4. Indicators

I will describe each of these concepts below.

THE QUALITY MODEL

The SQALE quality model is the list of good practices that a project team or organization considers its definition of "right code." This list will serve as a reference for estimating the technical debt of the code. Any noncompliance with the quality model creates debt and, conversely, there is no debt without the breach of at least one of the requirements.

If you don't have such a list and don't have time to establish one, you can use the Agile Alliance Debt Analysis Model (A2DAM) just released by the Agile Alliance. The A2DAM is a list of very basic good practices that you can use as a quick start. You can also use the default list provided by your static analysis tool. Project retrospectives are good opportunities to adapt and enrich the initial list to the specific context of your project.

THE ESTIMATION MODELS

The SQALE method contains two estimation models. One is used to estimate the time to remediate each debt item contained within the code and identified by the static analysis tool. This time is the principal associated with the debt item and is called the *remediation cost*. As an example, during the last week, a project team made some "quick-and-dirty" implementations in order to satisfy an important deadline. In doing so, they made 15 violations of their definition of right code. Using the SQALE estimation model, the tool will estimate the associated remediation cost as 3h 20min. In other words, by taking some short cuts, the team has "borrowed" 3h and 20min of work, time that they will have to spend later to implement the code correctly.

The second model estimates the impact of the debt items on the business and is called the *non-remediation cost*. It estimates the future additional costs, such as extra work imposed on anyone working with the code, that arise from technical debt. This cost could also be considered as the cost of delaying the remediation.

Therefore, with SQALE, each debt item has two costs: the remediation cost and the non-remediation cost. All these calculations are performed by the analysis tools supporting the method. Most of these tools have

©2016 Cutter Information LLC

SQALE estimation models already preconfigured, so you can start analyzing your code with their default settings. As with all estimation models, it will be beneficial to adjust them after you have seen the results.

When you add all the remediation costs of all the debt items discovered by the code analysis, you get the *technical debt* of the component, application, or software domain. When you add all the non-remediation costs of a software component, you get the *business impact* (the interest part) of the component.

THE INDICES

I have already introduced the technical debt index and the business impact index resulting from the two estimation models of the method. These two indicators should be monitored and made transparent to all the participants in a project. Everybody will know at all times how much technical debt the project is facing in terms of either principal or interest.

Another important index is the SQALE debt ratio, which is the technical debt divided by the budget of the project. To get back to the financial metaphor, one way to evaluate the health of a company is to calculate its debt ratio, which is the ratio between the company's debts and assets. By analogy, the SQALE debt ratio allows you to monitor the health of your projects and applications.

THE INDICATORS

The most used indicator is the SQALE rating (see Figure 1).

It is obtained by plotting the SQALE debt ratios of projects and applications on a grid in order to yield a simple letter grade: A, B, C, D, or E (see Figure 2). A low debt ratio produces an A grade, whereas a high debt ratio results in a E grade. Associating colors to each grade allows the tool to present a global map of a very large portfolio, thus enabling users to immediately identify a potential threat (see Figure 3[4]).

The second most used indicator is the SQALE pyramid, which represents the distribution of technical debt in terms of quality characteristics (see Figure 4). This indicator can be read in two ways:

1. The analytic view (represented by the numbers in the left column and the light blue bars)
2. The consolidated view (represented by the numbers in the right column and the dark blue bars)

Let's start with the analytic view. In the example shown in Figure 4, the debt related to reliability is 8d 1h. If this amount is perceived as more debt than is desirable, the development team can initiate training or coaching on one or more factors that are the cause of this debt (e.g., inadequate exception handling, or a potential null pointer exception). By taking these steps, the team can limit the rate of future technical debt accumulation and improve the reliability of the delivered code.

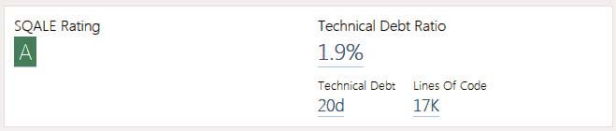

Figure 1 — An application summary in SonarQube.

Rating	Up to	Color
A	1%	
B	2%	
C	4%	
D	8%	
E	∞	

Figure 2 — An example of a SQALE rating grid.

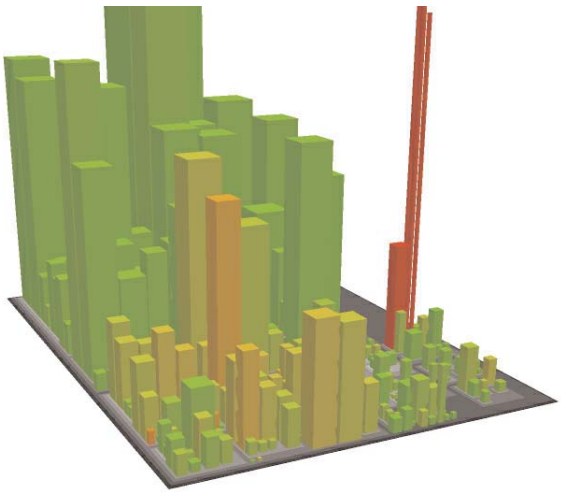

Figure 3 — A city-like representation of a large portfolio.

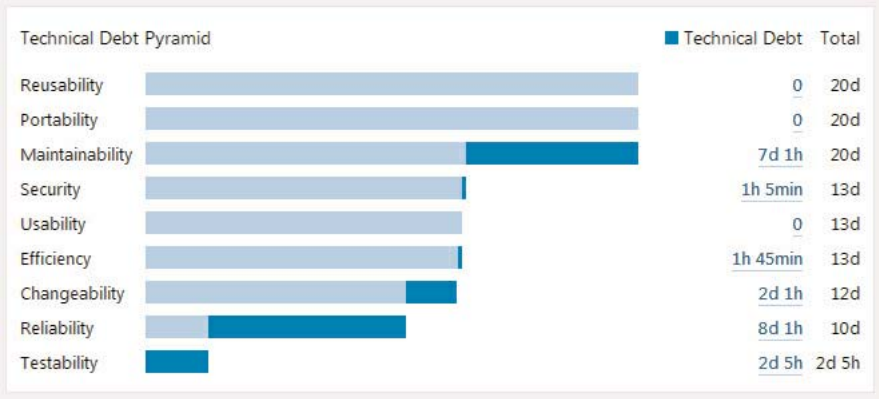

Figure 4 — A SQALE pyramid.

Now let's see how to interpret the consolidated view of the pyramid, which is obtained when we add the debt of all lower characteristic levels for a given characteristic. These calculations are shown by the numbers in the right columns. Take the example of changeability, which has a technical debt of 12 days. Agile projects generate a large number of change cycles to the code. The necessary quality characteristics to support these developments are testability, reliability, and changeability. Because changeability builds on reliability and testability, the true distance between the current state of the code and the target state of having easily changeable code is the summation of the debt associated with each of the three characteristics (2d 5h + 8d 1h + 2d 1h = 12d 7 h [rounded to 12d by the tool]). This consolidated value answers the following question from a business representative: "How far are we from having changeable software?" This consolidation mechanism is applicable to all characteristics.

The last SQALE indicator I want to introduce is the SQALE debt map. This is a bubble graph on which an item (a file, a component, an application) is represented on two axes, the technical debt and the business impact (see Figure 5). I will discuss its usage later on.

MANAGING THE TECHNICAL DEBT OF A PROJECT OR AN APPLICATION

Once you know how much debt you are facing and you have the ability to analyze the nature of the debt, you are in quite a good position to start paying your debt back.

The first instinct would be to rush in and fix the debt items that have a very high impact (non-remediation cost) and a low remediation cost. Such remediations will have a very high return on investment and so they will be easy to justify.

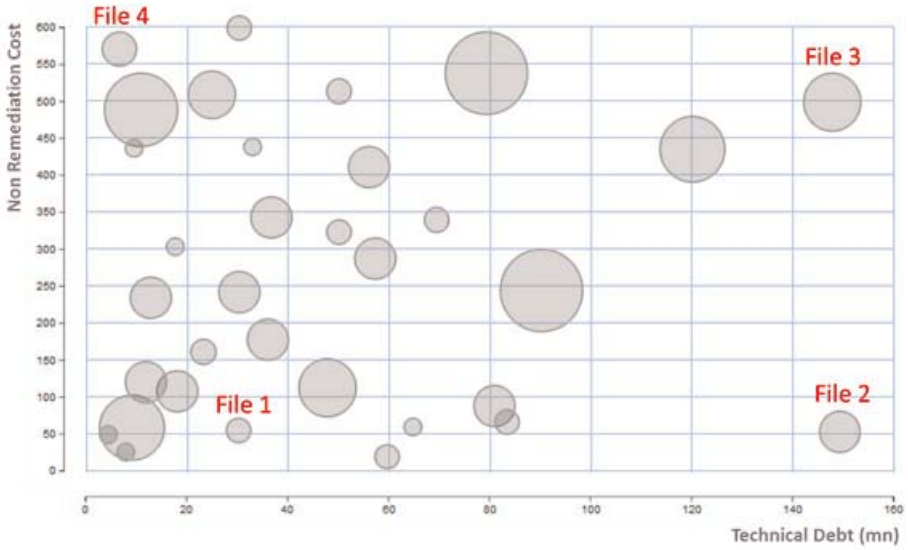

Figure 5 — A SQALE debt map at the file level.

©2016 Cutter Information LLC

While this logic sounds rational, in many cases it is not optimal. In reality, you will find that some debt items, including some with very high impact, are located in pieces of code that also have a structural debt issue. These pieces should be completely refactored because they are too complex, have bad coupling, or appear to be duplicated code. In such cases, if you start fixing the high-impact issues first, the time you spend on those items will be lost when you fix the structural issues later. There are frequent scope dependencies between debt items, and this should be taken into account in the prioritization.

So, because the prioritization of refactoring is more complex than it appears at first glance, the SQALE method supports three different strategies that correspond to different contexts. The key input is the available budget for the refactoring.

Case 1

You are far from the delivery date, and you are able to allocate time to address a large percentage of your total technical debt (at least 60%).

In this case, as I explained before, you need to improve the quality of code by first fixing the structural debt. In practice, this is equivalent to making it testable. This is the technical debt associated with the first level of the SQALE pyramid, and it relates to issues such as too complex methods, duplicated code, and so on. After that, you pay back the debt associated with the next layer of the SQALE pyramid, which is reliability. And you continue up to the highest level of your pyramid.

Case 2

You have limited time. You can't repay the debt related to testability because it's structural and too time-consuming. You will be constrained to deliver your application with remaining debt. Thus it would be wise to reduce the business impact (or the non-remediation cost) of this debt. You will focus your efforts on fixing the issues with the highest potential business impact. These are, in general, the issues related to reliability and security. In the case of the example in Figure 4, you will need about 8d and 2h.

Case 3

You have very limited time. You don't have the time to pay back all the debt associated with reliability and security, so you focus on the remediations that have the highest return on investment. In this case, you use the debt map (Figure 5), and you fix files in the upper left corner, because their fixes have a very high ROI.

As noted earlier, this last strategy to improve the code is not optimal, because you will probably fix potential bugs in pieces of code that should be refactored for structural reasons. If so, this time will be lost. We can say that this is the quick-and-dirty way to manage technical debt.

These three cases summarize how the SQALE method provides relevant indicators that help teams and other interested parties make the optimal debt remediation decision, whatever the context.

If an application provides very little business value and its annual maintenance workload is very low, the fact that it is not well positioned in the debt map is not worrisome.

MANAGING THE TECHNICAL DEBT OF A LEGACY PORTFOLIO

From a management point of view, it is very beneficial to get a transparent view of the technical debt of the complete portfolio. To do so, the SQALE method proposes using a portfolio-level debt map. In this case, the points on the map are applications. Each application is positioned according to its technical debt density and its non-remediation cost density. This allows teams, portfolio managers, executives, and others to analyze the situation of the complete portfolio and to compare all the different applications whatever their technology, size, or context.

Consider the example shown in Figure 6. App B contains about 5 times more technical debt than App A. All things being equal, the technical debt of App C is 40 times more dangerous than that of App A. Using the debt map in this way will help you to analyze the situation and identify which part of your portfolio needs attention.

If an application provides very little business value and its annual maintenance workload is very low, the fact that it is not well positioned in the debt map (meaning that it is in the top right corner) is not worrisome. On the contrary, if an application is critical and its code is not of good quality (i.e., also positioned at the top right of the debt map), this represents a risk, and paying back the technical debt of this application may be a high priority.

In order to get such visibility and decision capability, it is important to use consistent estimation models across the whole organization. This is not so easy to achieve,

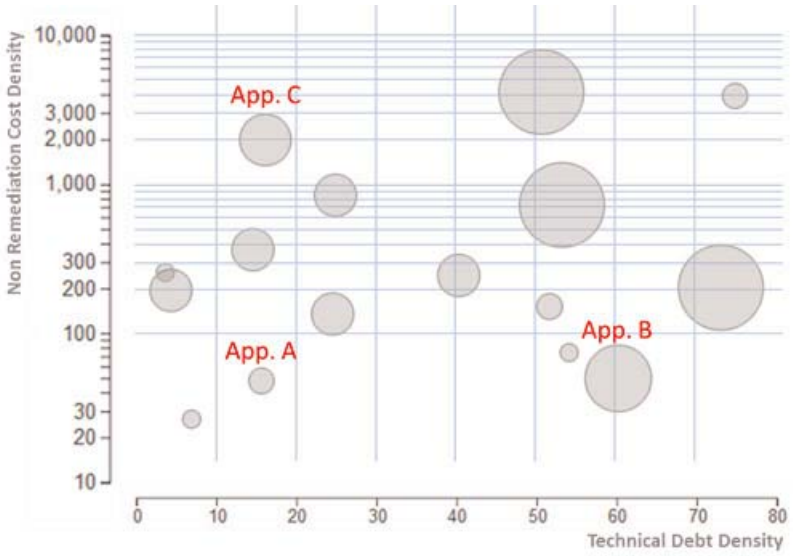

Figure 6 — A SQALE debt map at the application level.

because each business unit will find good reasons for using its own models. Establishing common estimation models is a corporate initiative and should be sponsored at the C level. It may require some external consulting support to achieve consensus among all the business units.

PARTING THOUGHTS

In addition to the debt ratio and the rating, the SQALE method includes many more useful indicators that provide deep insights into an organization's technical debt situation and support optimal decisions about it. This set of graphic indicators helps to establish a visual language for reporting the current state of a project or portfolio. This visual language is tool independent, which is an important contribution to the success of the method.

ENDNOTES

[1]The method definition document and other related articles and blog posts are available at www.sqale.org.

[2]The SonarQube platform is an open source project powered by SonarSource. Most of the graphics samples provided in this article are screenshot-produced with this tool.

[3]Campbell, G. Ann. "Mainstream: Noun. The Principal or Dominant Course, Tendency, or Trend." SonarQube, 30 September 2015 (www.sonarqube.org/mainstream-noun-the-principal-or-dominant-course-tendency-or-trend).

[4]The map provided in Figure 3 is produced by a SonarQube plugin developed by Excentia (www.excentia.es).

Jean-Louis Letouzey is an expert consultant at Inspearit and the author of the SQALE method for managing technical debt. Mr. Letouzey consults to the world's leading organizations for the implementation of corporate solutions for managing technical debt. He is also frequently embedded in due diligence teams for assessing the technical debt of the acquired software.Mr. Letouzey is a frequent technical speaker at international conferences. He can be reached at jl.letouzey@gmail.com.

©2016 Cutter Information LLC