SQALE - Technical Debt

by
Ramraj S ME.,
Assistant Professor
Department of Software Engineering

30 -Jan-2017

Agenda

- What is Techincal Debt
- Technical Debt Measurement
- Technical Debt Overview
- SQALE
- SQALE Overview
- Remidiation and Non-Remidiation

What is Tech debt

The term technical debt refers to delayed tasks and immature artifacts that constitute a "debt" because they incur extra costs in the future in the form of increased cost of change during evolution and maintenance. Technical Debt and Interest on technical debt are not same

1. The term technical debt is defined as the cost to improve technical quality up to a level that is considered ideal.
2. The interest of technical debt is the extra cost spent on maintaining software as a result of poor technical quality.

Technical Debt Measurement

The cost of technical debt can be broken down three ways as follows:

- 1. Principal. This is the amount of effort in dollars that it would take to fully service the technical debt.
- 2. Recurring interest. This is the cost to the organiza- tion for holding onto the debt
- 3. Compounding interest. This is the additional technical debt that accrues over time.

Technical Debt Overview

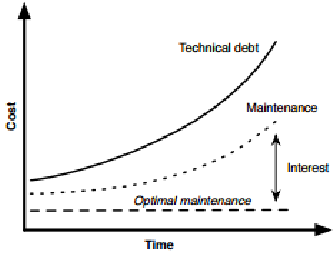


Figure 1: Technical debt and its interest grow over time if not resolved

Figure :

SQALE

- Technical debt measurement needs a quality framework to formulize quality of the system and the ideal quality.
- SQALE method (Software Quality Assessment based on Life Cycle Expectations)
- The debt evaluated with SQALE is the internal debt associated to the source code of an application. This excludes process related debt.
- The issues may come intentionally (as a way to achieve the objective of a sprint) or unintentionally. In both cases, there will be a negative impact (which means interest to pay), so this should be counted as debt by the method.

SQALE - Quality Framework or Model

Principles

- quality of the source code is a non-functional requirement
- Formalising requirements in relation to the quality of the source code
- Assessing the quality of a source code
- Cost remediation to meet the Quality
- assesses the importance of a non-conformity quality
- SQALE Methods Quality Model is orthogonal
- SQALE Method uses addition for aggregating the remediation costs, the non-remediation costs and for calculating its indicators

SQALE Overview

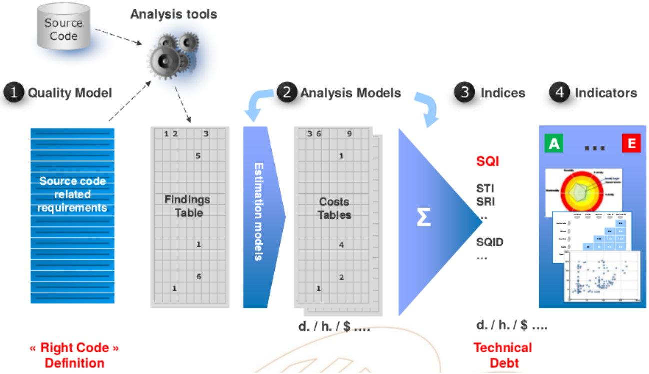


Figure :

SQALE

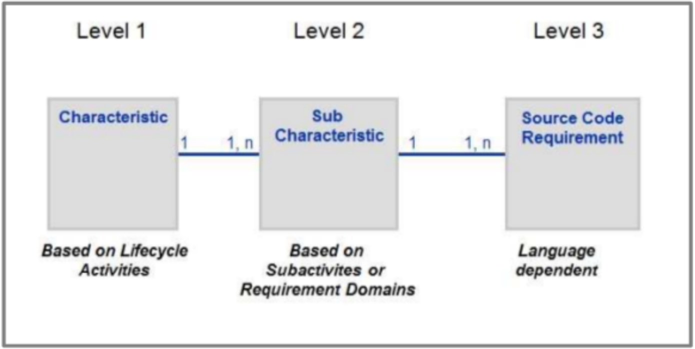


Figure : Classification

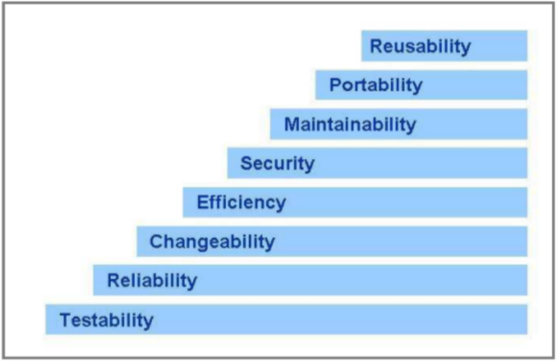


Figure : First Level

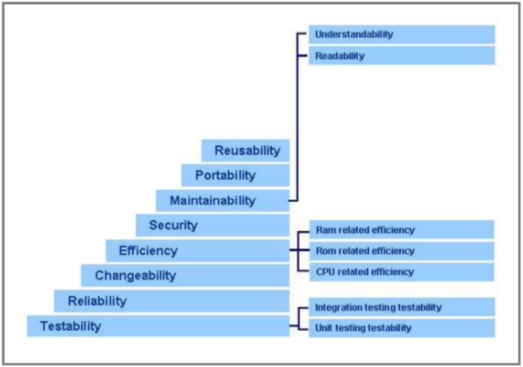
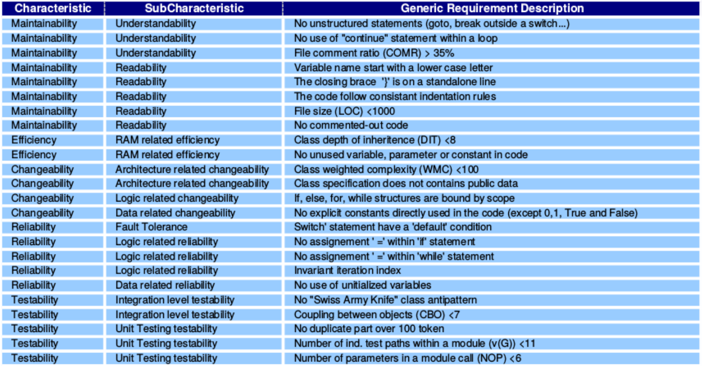


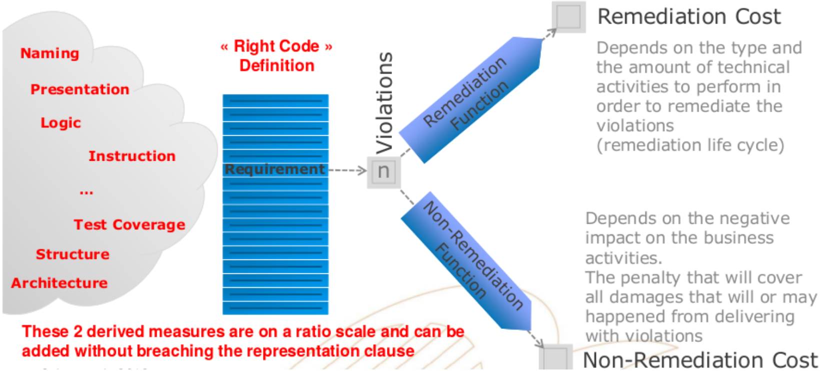
Figure : Second Level



Characteristic	SubCharacteristic	Generic Requirement Description
Maintainability	Understandability	No unstructured statements (goto, break outside a switch...)
Maintainability	Understandability	No use of "continue" statement within a loop
Maintainability	Understandability	File comment ratio (COMR) > 35%
Maintainability	Readability	Variable name start with a lower case letter
Maintainability	Readability	The closing brace '}' is on a standalone line
Maintainability	Readability	The code follow consistant indentation rules
Maintainability	Readability	File size (LOC) <1000
Maintainability	Readability	No commented-out code
Efficiency	RAM related efficiency	Class depth of inheritence (DIT) <8
Efficiency	RAM related efficiency	No unused variable, parameter or constant in code
Changeability	Architecture related changeability	Class weighted complexity (WMC) <100
Changeability	Architecture related changeability	Class specification does not contains public data
Changeability	Logic related changeability	If, else, for, while structures are bound by scope
Changeability	Data related changeability	No explicit constants directly used in the code (except 0,1, True and False)
Reliability	Fault Tolerance	Switch' statement have a 'default' condition
Reliability	Logic related reliability	No assignement ' =' within 'if' statement
Reliability	Logic related reliability	No assignement ' =' within 'while' statement
Reliability	Logic related reliability	Invariant iteration index
Reliability	Data related reliability	No use of unitialized variables
Testability	Integration level testability	No "Swiss Army Knife" class antipattern
Testability	Integration level testability	Coupling between objects (CBO) <7
Testability	Unit Testing testability	No duplicate part over 100 token
Testability	Unit Testing testability	Number of ind. test paths within a module (v(G)) <11
Testability	Unit Testing testability	Number of parameters in a module call (NOP) <6

Figure :

Remidiation and Non-Remidiation

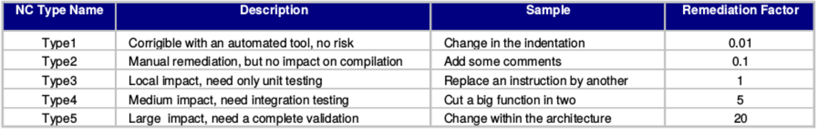


Depends on the type and the amount of technical activities to perform in order to remediate the violations (remediation life cycle)

Depends on the negative impact on the business activities.
The penalty that will cover all damages that will or may happened from delivering with violations

These 2 derived measures are on a ratio scale and can be added without breaching the representation clause

Figure :



NC Type Name	Description	Sample	Remediation Factor
Type1	Corrigible with an automated tool, no risk	Change in the indentation	0.01
Type2	Manual remediation, but no impact on compilation	Add some comments	0.1
Type3	Local impact, need only unit testing	Replace an instruction by another	1
Type4	Medium impact, need integration testing	Cut a big function in two	5
Type5	Large impact, need a complete validation	Change within the architecture	20

Figure :

Example of a Remediation Function for the Comment Ratio

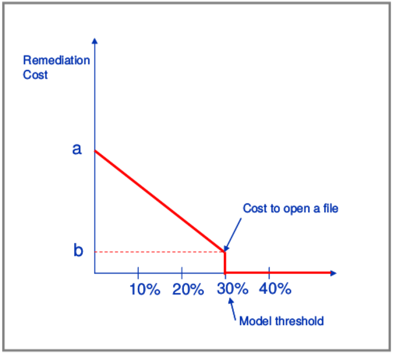


Figure :

Example of a Remediation Function for Cutting an Artefact

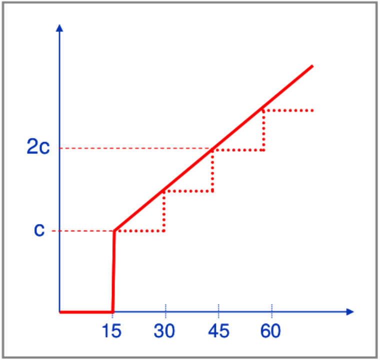


Figure :

References I

[1] Stephen Chin, Erik Huddleston, Walter Bodwell, and Israel Gat, *"The Economics of Technical Debt"*

[2] Jean-Louis Letouzey, *"The SQALE Method"*

Thank you