THE AGILE ALLIANCE DEBT ANALYSIS MODEL

By Jean-Pierre Fayolle, Thierry Coq and Jean-Louis Letouzey

Contributors: Declan Whelan, Tom Grant and Dan Sturtevant

As explained in our introduction document about technical debt, it is important to identify and analyze technical debt in order to take relevant decisions about its pay-back.

One of the first steps, which is the identification of technical debt item, could be tricky or at least time consuming. So, we have prepared a “ready to use” list to help agile teams to do so.

This list is the A2DAM (Agile Alliance Debt Analysis Model). It contains good practices which, when violated, generate technical debt.

One will find below:

- The main goal and usage of this list.
- How it has been built.
- Its content.
- Assumptions for the estimation model.
- Impacted characteristics.
- How to use it.
- Recommendation for extending it.

Main goals of A2DAM

This list responds to basic uses cases such as:

- Use Case 1: A new agile team is ready to start a project and is working on its Definition Of Done (DOD). They want to include criteria about amount and type of technical debt.
- Use Case 2: A computer programming teacher is looking for a list of simple and basic coding practices to introduce to his/her students.
- Use Case 3: An organization outsources a development project and would like to add source code related requirements. This to make sure that the delivered code will be maintainable in the future.
- Use Case 4: One is buying a company. One of the major assets is an application. During the due diligence, one wants to evaluate the technical debt of this software by using a “on the shield” and independent method.

The A2DAM list is a ground level list. Its size has been voluntary limited. It should be considered as a starting point. We encourage (see details below) mature teams or critical projects to enrich it to suit their context.

One of the goals of the A2DAM list is to be easily verifiable by automated tools. Good practices which can't be easily verifiable by tools have been intentionally discarded.

There is nothing new in it; its contents should be familiar to most of the developer community.

How it has been built

An initial version has been built by the AA Program Team which is formed by members with recognized experience either in agile development or in source code quality.

This initial list has been sent for review to experts in 16 companies developing static code analysis tools. 11 of them have provided their feedback about the clarity of the practices, the associated thresholds, their relevance and tool implementation. The list of experts is provided in an addendum.

The A2DAM list is available in 2 forms:

- The short form will suit for communication, training, inclusion within a DOD list.
- The detailed form provides details like Rationale, Remediation Cost and Type, traceability to coding principles, etc. (see below the complete list of fields associated to each practice).

Table content

Each practice is described by the following fields:

- **ID**: A prefix + 4 characters identifier for the practice.
- **Good practice**: A short description of the practice.
- **Ground level threshold:** Threshold of the measure (when applicable) for 'ground level' developers, based on assumptions like 'Developers have some coding experience, they are not novice, they are not champions, ...' or 'The developers or the team owns the code he/she works on' (see 'Assumptions' hereunder).
- **Main characteristic impacted**: Main characteristic for this practice. This characteristic is defined in the model provided hereafter.
- **Technology**: Technologies for which the practice applies. 'New technologies' are defined as: Languages usually found on Agile projects (Java, Javascript, C++, …). Often but not necessarily Object Oriented.
- **Rationale:** A short explanation about why a violation of the practice will incur technical debt.
- **Remediation:** Suggestion about way(s) to fix the violation.
- **Remediation Cost Type**: Describe the model proposed for estimating the debt associated to each debt item.

    - 'Fixed': 'Remediation Cost' is constant, i.e. the same for each violation to be fixed.
    - 'Gap dependent': 'Remediation Cost' is proportional to the difference between the measured value and the threshold parameter for this practice.
- **Remediation Cost**: The value(s) proposed for using the proposed debt estimation model.
- **Potential Impact Level**: Level of impact on the business when the practice is violated.
- **Remediation Scope**: Impact scope of the remediation within the code: line, block, method, class, file, etc. In most cases, it gives an idea of the amount and type of tests needed to verify the remediation.
- **SOLID**: When applicable, the SOLID characteristic associated to the practice. For instance, 'Single Responsibility Principle'.
- **Simple Design**: When applicable, the Simple Design characteristic associated to the practice: 'Tested', 'DRY', 'Clear', 'Concise'.

Assumptions of the estimation model

The A2DAM list provides a remediation cost estimation model associated to each practice. This allows estimating roughly the principal part of the technical debt of your application.

The provided estimation model is based on the following assumptions:

H1: Development is performed in an Agile context.

H2: The check in and check out time in a Configuration Management System (CMS) or a Control Version tool is negligible. *If your network or CMS is slow, you are facing another type of debt that you should manage.*

H3: The integration test workload is not included in the Remediation Cost estimation. *This may be automated and transparent or performed by another team.*

H4: Compilation time is not included in the Remediation Cost estimation. *Normally developers do something else during that time. If compilations are "time consuming", you are facing another type of debt that you should manage.*

H5: Remediation Cost covers: coding, unit testing development and debug, and regression testing update.

H6: Generated code should be excluded. *Evaluate your generated code only once and if needed, improve/fix your generator.*

H7: The development team uses the most appropriate IDE for the language. *If you don't, you are facing another type of debt that you should manage.*

H8: Developers have coding experience. They are not novices, nor are they expert coders but they know how to use their development environment effectively.

H9: Developers will understand the issue raised by the Source Code Analysis tool and the suggested remediation proposed. *Either the developer is enough experienced, either the tool provides some explicit guidance.*

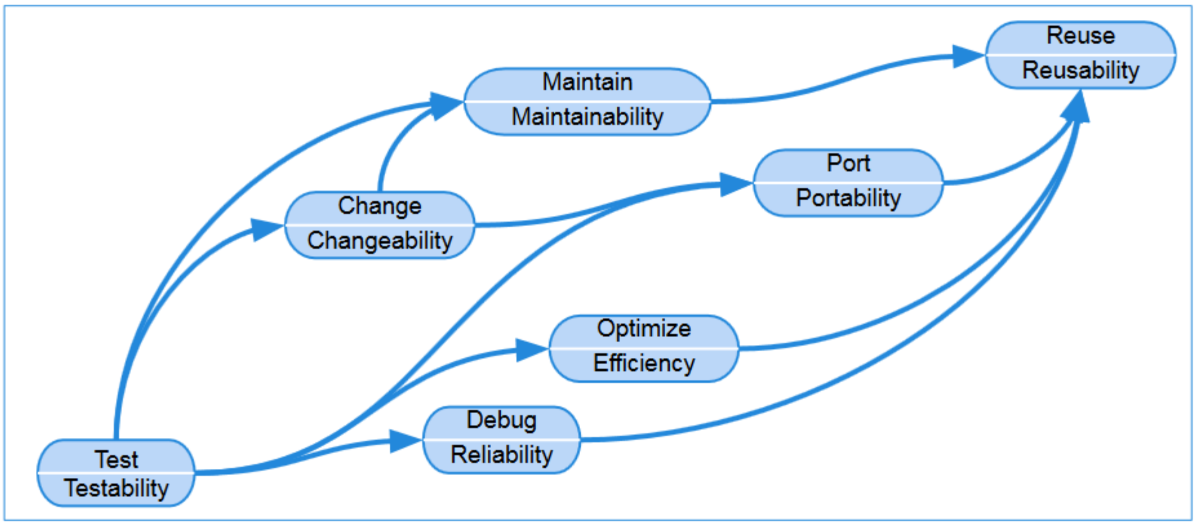
H10: Developers or the development team owns the code they work on. *The developer understands what he/she is programming (or will get a quick support from other team members).*

H11: The estimated Remediation Cost does not cover process specific tasks like updating UML models, technical documentation, reporting, etc. If needed, you should estimate such activities separately.

H12: All development, testing, etc. environments/tools are up to date and perform well. *If not, you are facing another type of debt that you need to manage.*

Impacted characteristic

As shown in the following diagram, they are numerous dependencies between all software quality characteristics. As an example, a practice violation that affects the testability of a file, will also indirectly affect its reliability, changeability, maintainability etc. So in our A2DAM table, we restrict the impact of a practice at the most direct impact and we don't list other indirect impacts.



How to use it

1) If you use the A2DAM in the context of the Use Case 1 listed previously

The first step is to set criteria for the DOD of your project. You may have a DOD at sprint level and another one at release level. You use the A2DAM list for your list of practices that generate technical debt when they are violated and as a model to estimate the total amount of technical debt of your software.

We suggest then to set 2 separate criteria:

- One regarding the amount of technical debt. This should be used to limit the principal of the technical debt. This should be established as either as a density per new lines of code, or a ratio between your technical debt and the time spent on the project. As example:
    - Sprint level DOD criteria : 10 % (technical debt/dev time).

    - Release level DOD criteria: 5%.
- One regarding the nature, the type of technical debt. This should be used to limit the impact of the technical debt. As an example:
    - No violation incurring “Very high” impact.

As said before, one of the goals of the A2DAM list is to be easily verifiable by automated tools.

The second step is to implement the list and the estimation model within an automated solution to make sure that the identification and analysis of technical debt could be performed as much as needed and does not require specific effort. Make sure that the technical debt analysis results are available to every stakeholder of your project. Good visibility and good analysis capabilities of the Technical Debt are key success factor. Use the “Potential Impact Level” and “Main characteristic impacted” attributes provided into the A2DAM to analyze the distribution of your technical debt.

From that point, agile principles and practices will ensure that the relevant decisions regarding the technical debt will be performed within your context. Refer to our document “Project Management and technical debt” to get suggestions at Release, Iteration and Story levels.

2) If you use the A2DAM in the context of Use Case 2 listed previously

Start to introduce the “SOLID” and “Simple Design” principles. Then present the main attributes provided with each practices of the A2DAM. Finally, present the list of practices and explain some of them with real code to make the presentation more concrete.

3) If you use the A2DAM in the context of Use Case 3 listed previously

Include the A2DAM into the contract with your outsourcer. Set associated goals for the code to be developed. As example:

- Acceptable technical debt ratio: < 5%.
- Acceptable impact of the technical debt: No violation with “Very high” impact.

4) If you use the A2DAM in the context of Use Case 4 listed previously

Analyze the software to be acquired using the A2DAM.

- If the resulting technical debt ratio is over 15%, it is likely that the code will be difficult to change and to maintain.
- If the resulting technical debt ratio is over 30%, it is likely that the code will be more a liability than an asset.

You may complete your analysis with other information like:

- Amount of technical debt associated to “Very High” impact violations.
- Amount of technical debt associated to Testability related violations.

How to extend it

During each retrospective of your project, one will find opportunities to improve the initial list by:

- Adding practices into one's Definition of Done.
- Tailoring the estimation model based on real feedback data.
- Changing practice thresholds.
- Removing practices which are not relevant into your project's context.

When one's project has high level expectation regarding characteristics like reliability or security, it will be especially relevant to add dedicated practices to better cover this specific goals.
For such situations, you can find help with more complete and more dedicated lists like the MISRA rules (dedicated to the automotive market) and the CERT rules (dedicated to security).

Acknowledgements

The Agile Alliance group 'Managing the Technical Debt' would like to thank the reviewers from software vendors who collaborated to the edition of the Agile Alliance Debt Analysis Model

BUGSENG / Reviewer: Roberto Bagnara

Buguroo / Reviewer: Isaac Cruz

Cast software / Reviewer: Luc Béasse

GrammaTech / Reviewer Roger Scott

Hello2morrow / Reviewer: Alexander von Zitzewitz

NDepend / Reviewer: Patrick Smacchia

Optimyth / Reviewer: Juan Pablo Tejela

Rogue Wave Software / Reviewer : Mike Bessuille

Security Reviewer Limited / Reviewer: Tim Moses

SonarSource / Reviewer: G. Ann Campbell

Squoring / Reviewer: Patrick Artola