ATDx: Prototype Implementation Technical Report

1 ATDx APPROACH VIABILITY

In this technical report we document the investigation carried out to evaluate the viability and the implementation feasibility of the proposed *ATDx* approach. This process is carried out in order to get further insight into the potential benefits and drawbacks entailed with the concrete implementation of an *ATDx* instance. Specifically, we apply the *ATDx* methodology to the SonarQube source code static analysis tool and a large-scale set of open source Java-based projects.

2 ATDX BUILDING STEPS EXECUTION

In the remainder of this section we document the process followed to build an *ATDx* prototype by following the implementation steps defined in the approach.

2.1 Step 1: *AR* Identification from SonarQube Rule Set

The first step of the *ATDx* establishment entails the identification of a set of architectural rules AR^T for a tool T. To this aim, we selected as our initial set of source-code rules R^{SQ}, the SonarQube design rules presented by Ernst et al. [Ernst et al., 2017] and specific to the Java programming language. Our rationale is multi-fold: (i) the rules focus on software design, and hence they may be good candidates for having architectural relevance, (ii) the rules are implemented in a prominent static source-code analysis tool, easing the $AR_i^{SQ}(SUA)$ measurement retrieval process, and (iii) given the widespread adoption of SonarQube and Java in industrial contexts [Janes et al., 2017], the experiment results can be leveraged in follow-up studies with industrial parties.

Given that the rules considered are implemented in SonarQube, we knew *a priori* that they can be associated to a granularity level Gr^{SQ}, that each can be considered as a function $AR_I^{SQ} : [E] \to \{0,1\}$, and that it is possible to retrieve via source-code analysis the granularity Gr_i^{SQ} associated to each rule *i*.

To evaluate the two criteria which define an analysis rules R^T, as *architectural rules* AR^T, we carried out a manual inspection of the definition of each rule, available in the official documentation of the static analyzer[1]. The selection process was executed by analyzing the content of each rule description, and evaluating it against each criterion.

To mitigate potential threats to construct validity, two independent researchers carried out the identification of the AR^{SQ} rules on the predefined set R^{SQ} of Java-based rules in SonarQube. A third researcher with several years of experience in software engineering was involved to resolve potential conflicts and review the AR^{SQ} identification results. In a first iteration a 72.2% of agreement was reached, which then was resolved with the intervention of the third researcher.

From the initial set of 72 SonarQube design rules presented by Ernst et al. [Ernst et al., 2017], we identified 45 architectural rules. The full list of all rules supported by SonarQube and the identified subset of architectural rules are included in the replication package of this study.

2.2 Step 2: Formulation of the Java-based 3-tuples $\langle AR_i^{SQ}, Gr_i^{SQ}, ATDD_j^{SQ} \rangle$

Once we established a set of AR^{SQ}, we proceeded to formulate the 3-tuples $\langle AR_i^{SQ}, Gr_i^{SQ}, ATDD_j^{SQ} \rangle$. This process is carried out by following the *ATDx* process defined to identify the 3-tuples, leading to the mapping for each AR_i^{SQ} its corresponding granularity Gr_i^{SQ}, and its ATD dimensions $ATDD_j^{SQ}$.

As for the identification of AR^{SQ}, we followed the previously presented research methodology carried out by three researchers in collaboration.

Regarding the granularity levels Gr^{SQ}, we identified 4 levels of granularity that characterized the rules in AR^{SQ}, namely *Java non-comment lines of code* (NCLOC), *Java method*, *Java class*, and *Java file*.

As for ATD dimensions $ATDD^{SQ}$, we observed the emergence of 6 core dimensions, namely *Inheritance*, *Exception*, *Java Virtual Machine Smell* (JVMS), *Threading*, *Interface*, and *Complexity*. The *Inheritance* dimension (9 rules) clusters rules evalu-

[1] https://docs.sonarqube.org/latest/user-guide/rules/

ating inheritance mechanisms between classes, such as overrides and inheritance of methods or fields. The *Exception* $ATDD^{SQ}$ (8 rules) groups rules relating to the Java throwable class "Exception" and its subclasses. *JVMS* (7 rules) embodies instead rules which assess potential misuse of the Java Virtual Machine, e.g., the incorrect usage of the specific Java class "Serializable". Rules associated with the *Threading* dimension (7 rules) deal with the potential issues arising from the implementation of multiple execution threads, which could potentially lead to concurrency problems. The *Interface* dimension (7 rules) encompassess rules assessing fallacies related to the usage of Java interfaces. Finally, the *Complexity* dimension (6 rules) encompasses rules derived from prominent complexity measures, such as McCabe's cyclomatic complexity [McCabe, 1976].

2.3 Step 3: Dataset establishment via SonarCloud

In order to build our dataset of $AR^{SQ}(SUA)$ measurements, we took advantage of the SonarCloud platform[2], which provides a dataset of pre-computed SonarQube analysis results of open-source software projects. Specifically, SonarCloud encompasses SonarQube analysis results of over 90K software projects, implemented in different programming languages, ranging from Python, to C++, and Swift.

As the starting step to build our *ATDx* analysis dataset, we selected from the SonarCloud dataset exclusively the software projects implemented in Java (as we base our *ATDx* analysis on Java-based AR^{SQ}, whose selection is documented in Section 2.1). This step led us to the identification of a preliminary set of 14,037 projects.

Next, we carried out a series of quality-filtering steps to ensure a good level of quality of the software projects considered for inclusion in our dataset. Specifically, a first filtering step removed from the dataset all the projects deemed as too small in order to be considered as "real-world" software projects. To this aim, we filtered out all software projects containing less than 100 Java NCLOC and 9 Java source code files. This process led to the removal of 4,431 projects.

We then sanitized our dataset by removing all software outlier projects according to one or more of the identified granularities Gr^{SQ} (i.e., Java NCLOC, Java method, Java class, and Java file). This process removed 1,801 additional projects, resulting in a preliminary dataset of 7,805 projects.

Finally, in order to avoid "toy" projects, we carried out an inspection of the project name of the remaining codebases. This step was carried out by identifying all the projects which contained one or more of the following keywords as sub-strings of their names: "demo", "course", "thesis", "exam", "tool", "util", "helper", "plugin", "plug-in", "homework", "sonarcloud", and "sonarqube". This led to the identification of 511 potential "toy" projects for potential exclusion. The names of the projects where then manually inspected by two independent researchers and, after reaching a consensus, the projects deemed as "toy" projects were removed from the dataset. This process led to removing 499 additional projects, leading to a final number of 6,706 projects as input to our *ATDx* experimentation. The metadata related to the projects either excluded or included at each intermediate filtering step is available in the replication package of this study.

After the identification of the dataset of projects, the SonarCloud API was leveraged in order to obtain the $AR^{SQ}(S)$ values for each selected project.

2.4 Step 4: *ATDx* Analysis Execution and Refinements

After we gathered the data necessary to carry out the *ATDx* analysis, we applied the *ATDx* analysis process on the established dataset of projects. We implemented *ATDx* via a script implemented in the R programming language[3].

While the values of $ATDD^{SQ}(S)$ and $ATDx^{SQ}(S)$ are, by definition, normalized in a range of $[0,1]$, we opted to further process such values by normalizing them within a range of $[0,5]$. While such operation does not entail any statistical change, we opted for such adjustment in order to ease the interpretation of such metrics by practitioners and researchers alike, facilitating the inspection of the *ATDx* analysis results, while ensuring their correctness.

Additionally, driven by preliminary inspection of the *ATDx* analysis results, we opted to apply a 1K multiplier to the intermediate *ATDx* value $NORM^{SQ}(S)$, in order to make such values more intuitive to interpret by a human reader.[4]

2.5 Step 5: Results Inspection

After executing the *ATDx* analysis, we conducted an inspection of the gathered results. Specifically, we conducted a statistical analysis on the calculated

[2] `https://sonarcloud.io/about`

[3] `www.r-project.org/about.html`

[4] This heuristic is bound to the experimental setting used, i.e. the identified architectural rules AR^{SQ}, their associated granularity Gr^{SQ}, and the dataset of projects we considered. Hence such heuristic may vary according to the experimental setting considered.

ATDx data in order to identify the most recurrent types of *ATDD*s. Additionally, we investigated to what extent the different *ATDD* dimensions are correlated, in order to identify if issues in one *ATDD* dimension may statistically influence also other dimensions. Finally, we investigated the *ATDx* analysis results by considering individually some prominent projects *P*, in order to gain further insights into the calculated *ATDD*^{SQ} values at project level.

3 APPROACH VIABILITY RESULTS

In this section we report the results of our *ATDx* viability investigation. Specifically, we inspect (i) the distribution of *ATDD* across projects, (ii) the correlation between *ATDD* dimensions, and (iii) the *ATDD* values of a subset of selected software projects.

3.1 *ATDD* distribution

We examine the distribution of *ATDD*^{SQ} across the software projects included in our dataset in order to get insights into the constituent parts of this specific *ATDx* instantiation. An overview of the *ATDD*^{SQ} value distribution, as well as the cumulative value of *ATDx*, obtained by calculating for each project *P* the mean value of its *ATDD*^{SQ}, is depicted in Fig. 1. Additionally, we report summary statistics of the *ATDD* dimensions in Table 1.

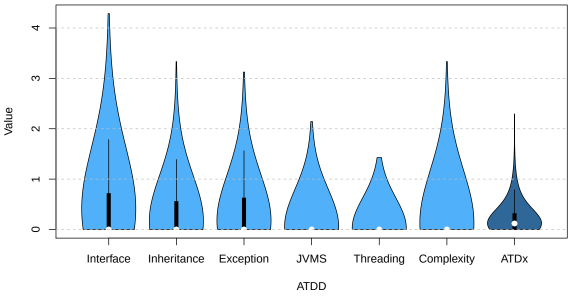


Figure 1: Distribution *ATDD*^{SQ} values across projects, and cumulative *ATDx*^{SQ} value

From the values reported in Fig. 1 and Table 1, we observe that all *ATDD*^{SQ} values are characterized by a median value of zero across software projects. This has to be attributed to the design and purpose of *ATDx*, which leverages outliers detection to identify severe ATD issues. Additionally, our analysis reveals that the overall *ATDx*^{SQ} has a median value greater than zero, indicating that more than half of the considered projects presents ATD issues in at least one of the *ATTD*^{SQ} dimensions. In fact, 4,179 out of 6,706 projects are characterized by ATD issues at a varying level of severeness. Regarding recurrence of *ATDD*^{SQ} dimensions, *Interface* is the ATD dimension which most frequently present issues across projects (3204/6706), followed by *Exception* (1863/6706), and *Inheritance* (1860/6706). Problems related to the *Complexity*, *JVMS*, and *Threading* dimensions are overall less frequent (1130/6706, 567/6706, and 272/6706, respectively).

Table 1: Descriptive statistics for the values of each *ATDD*^{SQ} dimension per project (SD = standard deviation, CV = coefficient of variation)

ATDD	Min.	Max.	Median	Mean	SD	CV
Interface	0	4.28	0	0.50	0.69	1.39
Inheritance	0	3.33	0	0.21	0.41	1.95
Exception	0	3.12	0	0.24	0.45	1.87
JVMS	0	2.14	0	0.06	0.23	0.38
Threading	0	1.44	0	0.03	0.15	5.0
Complexity	0	3.33	0	0.15	0.37	2.4
ATDx	0	2.29	0.11	0.15	0.26	1.7

3.2 *ATDD* dimensions correlation

After the analysis of the constituent dimensions of *ATDx*^{SQ}, we investigate if any statistically significant correlation can be observed between them. The execution of the Kruskal-Wallis omnibus test [Kruskal and Wallis, 1952] reveals that the distributions of *ATDD*^{SQ} values significantly differ, i.e., the mean ranks of the *ATDD*^{SQ} values across ATD dimensions is not statistically the same (*p-value* $< 2.2^{-16}$).

In order to get further insights into the correlation of *ATDD*^{SQ} values, we calculate the Spearman's rank correlation coefficient [Spearman, 1961] for the dimensions composing *ATDx*^{SQ}, in order to assess if any statistical relationship is established between them[5]. The results of such analysis are visually presented in Fig. 2.

From the correlation analysis emerges that no strong linear relationship can be observed across *ATDD*^{SQ} dimensions. Moderate correlations are instead present between the dimensions *Inheritance* and *JVMS* ($\rho = 0.54$), *Complexity* and *Interface* ($\rho = 0.45$), *Inheritance* and *Interface* ($\rho = 0.39$), and *Exception* and *Interface* ($\rho = 0.37$). The moderate correlation between those *ATDD*^{SQ} dimensions can be partially attributed to the underlying *AR*^{SQ} which the different dimensions share. Nevertheless, from an inspection of the mapping of *AR*^{SQ} to *ATDD*^{SQ} we observed that the moderately-correlated dimensions do

[5] We adopt the Spearman's ρ coefficient as, while carrying out a thorough selection of the software projects to be included in our dataset, we do not assume any normal distribution of the *ATTD*^{SQ} values.

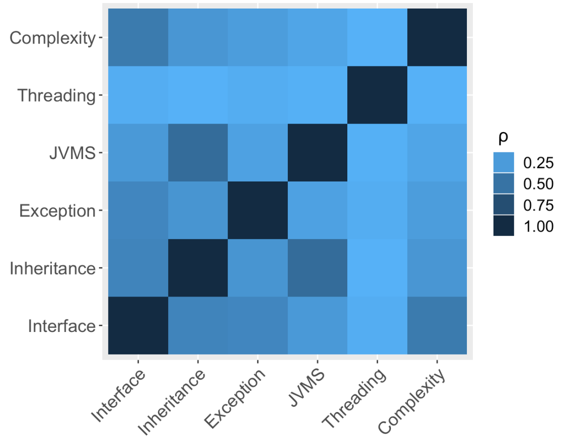


Figure 2: Spearman's rank correlation coefficient between architectural technical debt dimensions *ATDD*

not share AR^{SQ}. This leads us to conclude that the correlation is exclusively of a "data-driven" nature, rather than a "design-driven" one.

3.3 Inspection of $ATDD^{SQ}$ values of sample projects

In order to get further insights into our data at a refined level of granularity, we inspect the *ATDx* analysis results for a set of projects *P* by depicting their $ATTD^{SQ}(S)$ values via radar charts. The adoption of such graphical notation for displaying the normalized yet multivariate data composing *ATDx*, provides us with the means to intuitively compare our approach analysis results across different software projects, leading to what could be a general overview visualization of the *ATDx* results. Specifically, we carry out an *ad hoc* selection of the projects by identifying the ones which report the maximum value in one of the architectural technical debt dimensions $ATTD^{SQ}$. An overview of the $ATDD^{SQ}$ values calculated via *ATDx* for the selected projects is depicted in Fig. 3.

From Fig. 3 we observe a highly heterogeneous distribution of $ATDD^{SQ}$ values across the different projects. This finding highlights the lack of a strong correlation among the identified ATD dimensions, which was previously identified in our statistical correlation analysis (see Section 3.2). The visualization method reported in Fig. 3, in addition to the representation of $ATDD^{SQ}$ values, also provides a straightforward means to gain an intuitive picture of the overall ATD present in a software project analyzed via *ATDx*. In fact, we can intuitively evince that the area circumscribed by a radar chart is linearly proportional to the composite $ATDx^{SQ}$, i.e., the larger the area circumscribed by the radar chart of a project, the more severe is the overall *ATDx* of a project.

Interestingly, the project presenting the highest *Complexity ATDD* value (Fig. 3f), is also characterized by the highest composite value *ATDx*. By inspecting the radar chart, we can observe that this is mostly due to the high values of the *Complexity* and *Interface* dimensions, as the dimensions report lower values, and the *Threading* dimension is null.

The statistical findings regarding the *Threading* dimension, i.e., that such dimension is less frequent and characterized by overall lower values, can also be observed in the sample projects selected. In fact, 4 out of the 6 projects do not present any issues in this dimension.

Interestingly, in Fig. 3e, which represents the $ATDD^{SQ}$ values for the project with the highest *Threading* value, we observe that such project yields the lowest *ATDx* value among the projects considered. This fact, together with the previous observation on the *Threading* dimension and the high coefficient of variation reported in Table 1, points to some peculiar characteristic of this dimension. Nevertheless, an in-depth investigation should be carried out to verify this conjecture.

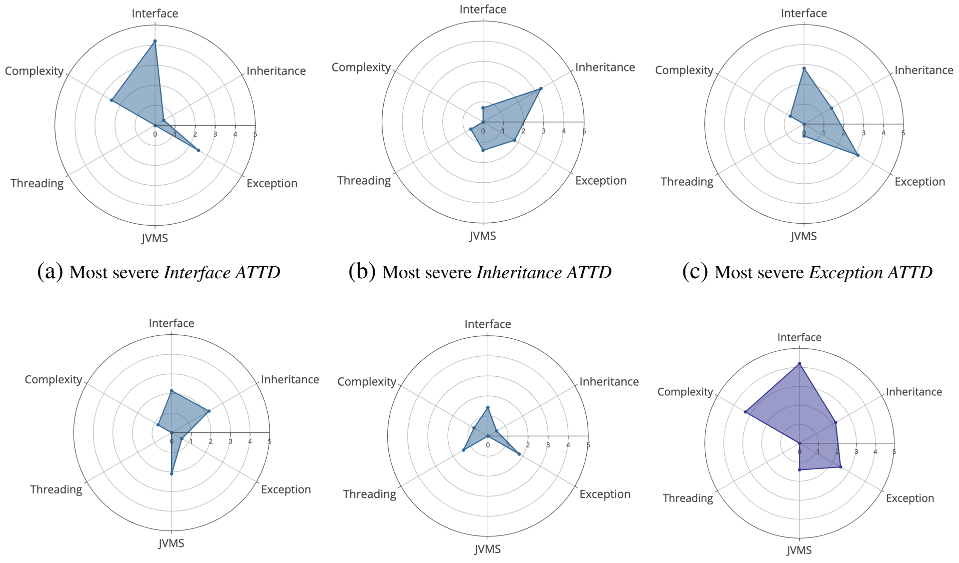
4 THREATS TO VALIDITY

4.1 External validity

It is important to stress that in this evaluation we are not claiming generalizability, but rather we aim at showing the viability and feasibility of the *ATDx* approach in the context of a tool widely use in practice and real-world Java projects. A potential threat to validity consists in the representatives of the projects selected in Step 3. We mitigated it by performing a series of quality-filtering steps to filter out corner cases or toy projects. Nevertheless, selected projects still exhibits a certain level of heterogeneity in terms of size and present architectural technical debt items. Moreover, SonarQube is one of the most used static analysis tools for Java-based systems [Janes et al., 2017], making us reasonably confident about the relevance of its rules in real-world projects.

4.2 Internal validity

The projects considered in the dataset provided by SonarCloud can also contain non-Java source code. To mitigate this potential threat to validity, we (i) consider only projects with a minimum number of Java artifacts and (ii) consider only SonarQube rules pertaining to Java. To avoid bias when selecting the AR^{SQ} rules, three researchers were involved in step 1 of the *ATDx* building process, their level of agreement was measured statistically, and disagreements were jointly discussed with the help of a third researcher. The same mitigation strategy was applied for the definition of the Java-based 3-tuples in step 2.



(a) Most severe *Interface ATTD* (b) Most severe *Inheritance ATTD* (c) Most severe *Exception ATTD*

(d) Most severe *JVMS ATTD* (e) Most severe *Threading ATTD* (f) Most severe *Complexity ATTD*. Also, most severe *ATDx* value.

Figure 3: *ATDx* results of software projects presenting a maximum value in one architectural technical debt dimension $ATDD^{SQ}$

4.3 Conclusion validity

We are reasonably confident of the correctness of the applied statistical methods; when inspecting the $ATDx^{SQ}$ and $ATDD^{SQ}$ values, we computed standard summary statistics on the obtained data and applied a non-parametric statistical test (i.e., Kruskal-Wallis) which does not make any assumption on the distribution of the analysed data. Also, a replication package with the raw data and analysis scripts is available for independent verification of the results[6].

4.4 Construct validity

It is important that the used analysis tool is configured correctly in order to provide reliable results. We mitigated this potential threat by reusing the dataset of analysis results provided by the SonarCloud platform. Moreover, our viability assessment suffers from a mono-method bias since we are consider only one analysis tool, i.e., SonarQube. This is in line with the feasibility-oriented nature of this assessment, which will be expanded to additional analysis tools in the next steps of this research line.

[6] https://github.com/ATDindeX/ATDx

REFERENCES

Ernst, N. A., Bellomo, S., Ozkaya, I., and Nord, R. L. (2017). What to fix? distinguishing between design and non-design rules in automated tools. In *IEEE International Conference on Software Architecture (ICSA)*, pages 165–168.

Janes, A., Lenarduzzi, V., and Stan, A. C. (2017). A continuous software quality monitoring approach for small and medium enterprises. In *8th ACM/SPEC on International Conference on Performance Engineering Companion*, pages 97–100.

Kruskal, W. H. and Wallis, A. (1952). Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, (47):583–621.

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, (4):308–320.

Spearman, C. (1961). The proof and measurement of association between two things. *The American Journal of Psychology*.