ATDx: A tool for Providing a Data-driven Overview of Architectural Technical Debt in Software-intensive Systems

Sebastian Ospina[1], Roberto Verdecchia[1], Ivano Malavolta[1] and Patricia Lago[1,2]

[1] *Vrije Universiteit Amsterdam, The Netherlands*
[2] *Chalmers University of Technology, Gothenburg, Sweden*

Abstract

Architectural technical debt (ATD) in software-intensive systems is mostly invisible to software developers, can be widespread throughout entire code-bases, and its remediation cost is often steep. In recent years, numerous approaches have been proposed to identify, keep track, and ultimately manage ATD. The variety of approaches available opens a new problem, namely how to gain an encompassing overview of the ATD identified in a software-intensive system. With this paper we make available the ATDx tool, an implementation of ATDx written in Python, designed in a plug-in fashion. ATDx is an approach designed to provide a data-driven, intuitive, and actionable overview of the ATD present in a portfolio of software projects. ATDx is based on third-party source code analysis tools, architectural issue severity calculation *via* clustering, and aggregation of measurements into different architectural technical debt dimensions. The ATDx tool allows users to automatically run the ATDx analysis, generate reports containing the ATDx analysis results, and is integrated with GitHub. In addition to the tool, we provide two already implemented plugins, allowing users to run the ATDx tool out-of-the-box.
GitHub repository: https://github.com/S2-group/ATDx
Video: https://youtu.be/ULT9fgxuB7E

Keywords

Software Engineering, Software Architecture, Technical Debt, Index, Tool

1. Introduction

Context. Architectural Technical Debt (ATD) in a software-intensive system encompasses all architectural design or implementation constructs that, while being suitable today, will lower the maintainability and evolvability of the system in the long term. ATD is mostly invisible to software developers, can be widespread throughout entire code-bases, and its remediation cost is often steep [1]. Tool support can aid software practitioners by providing automated and semi-automated approaches to systematically identify, keep track, and ultimately manage ATD. In recent years numerous fine-grained approaches, leveraging different definitions of ATD, have

ECSA'21: European Conference on Software Architecture, September 12–17, 2021, Virtual
s.ospinacamacho@vu.nl (S. Ospina); r.verdecchia@vu.nl (R. Verdecchia); i.malavolta@vu.nl (I. Malavolta); p.lago@vu.nl (P. Lago)
https://robertoverdecchia.github.io/ (R. Verdecchia); http://www.ivanomalavolta.com/ (I. Malavolta); http://patricialago.nl/ (P. Lago)
0000-0001-9206-6637 (R. Verdecchia); 0000-0001-5773-8346 (I. Malavolta); 0000-0002-2234-0845 (P. Lago)

been proposed to detect ATD [2]. The variety of ATD tools available, and the often fine-grained level of ATD identification results, opens a new problem, namely: *how to gain an encompassing overview of the ATD present in a software-intensive system?*
The Tool. This paper presents the ATDx tool, a Python-based implementation of ATDx [3]. The tool is available on GitHub[1]. ATDx is a data-driven approach that, by leveraging the analysis of a software portfolio, severity calculation of precomputed architectural violations *via* clustering, and severity aggregation into different ATD "dimensions" (ATDD), provides an overview of the ATD present in a software-intensive system. The ATDx tool implements the ATDx analysis, allowing users to execute the approach in a fully automated fashion. The tool is designed as configurable and extensible. The core functionalities of the ATDx tool are:

- **ATDx Execution:** The execution of all the automated steps required by ATDx, including architectural issue parsing, severity calculation *via* a clustering algorithm, aggregation of results into ATDD dimensions, and final ATDx index calculation.
- **Report Generation:** The automated generation of a report, summarizing the results of the ATDx analysis and providing additional insights.
- **GitHub Webhook**[2]: An integration with GitHub, enabling to automatically trigger the ATDx analysis when a GitHub event occurs (*e.g.,* a pull request[3]), and provide the results of ATDx as GitHub comments.

The ATDx analysis is by definition tool-independent, *i.e.,* it is not bound to any specific source code analysis tool as a source for detecting architectural rule (AR) violations. For this reason, the ATDx tool is designed in a plug-in fashion, implementing abstraction mechanisms that allow users to consider different sources of data, and further configure, customize, and extend the tool as needed.
Intended usage. The ATDx tool targets both researchers and practitioners as intended users. On one hand, researchers can leverage the implementation of ATDx to *independently conduct experimentation, replicate previous results, and extend or refine* the ATDx tool. On the other hand, practitioners can use the tool in their current practice, to *gain an overview of their ATD, track it, and tune the tool to best fit their preferences and context.*

2. The ATDx Tool

2.1. Architectural components

Figure 1 gives an overview of the architecture of the the ATDx tool. In the remainder of this section, we describe each component of the architecture.
FlaskServer. The ATDx tool is hosted on a Flask[4] Web server. While the tool can be run locally without the need of a dedicated Web server, the Flask server allows the ATDx tool to interact with GitHub repositories and make use of the GitHub Webhook functionality (see Section 1).

[1]https://github.com/S2-group/ATDx
[2]https://docs.github.com/en/developers/webhooks-and-events/webhooks/about-webhooks
[3]By default, pull requests are set in the ATDx tool as triggering events.
[4]https://flask.palletsprojects.com/en/2.0.x

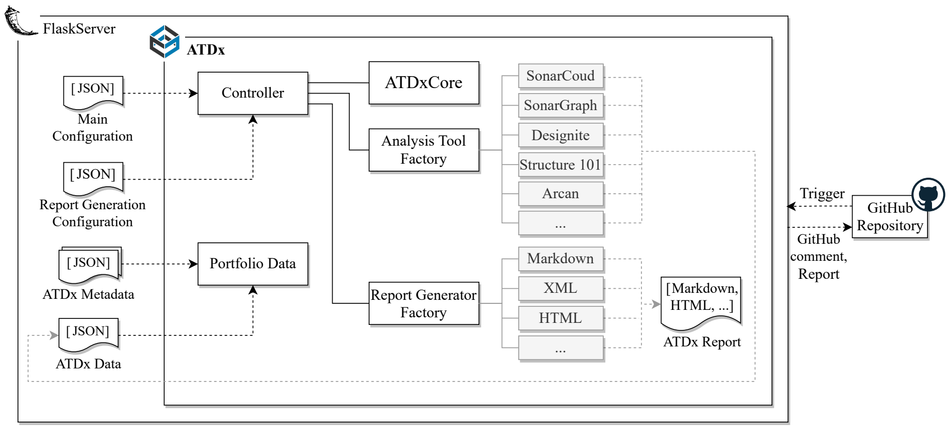


Figure 1: Architectural view of the ATDx tool

ATDxCore. This component implements the main business logic of the ATDx tool. Specifically, the component is responsible for executing the complete ATDx calculation, from the normalization of software metric measurements, to the severity calculation of AR violations, and calculation of ATDD and ATDx values (for more information on the ATDx approach, refer to the original publication [3]). The execution of ATDx can be invoked either by considering an entire software portfolio, or a single software project, if a dataset of a portfolio is already available.
Analysis Tool Factory. This component acts as abstraction mechanism interfacing the ATDx tool with the analysis tool(s) used to gather the AR violations used by ATDxCore (*e.g.,* SonarCloud, SonarGraph, etc.). This component implements the 'factory method' design pattern [4], where third-party analysis tools can be integrated as external plug-ins. This technical solution allows users to customize the ATDx tool according to their specific needs, while delegating the interfacing of their used analysis tool to the other components via the Analysis Tool Factory.
Report Generator Factory. The component Report Generator Factory is responsible for generating the report summarizing the ATDx results. Similarly to the Analysis Tool Factory, this component is implemented according to the factory method design pattern. This allows users to implement report generators which produce their preferred format (*e.g.,* Markdown, XML, etc.), without having to spend effort/time on how the created instance needs to interact with the other ATDx tool components. For more information on the information contained in the report refer to Section 2.2.
Portfolio Data. The component Portfolio Data acts as a centralized data storage. This component is used to store all the raw and processed data coming from, and used by, the other components[5]. The Portfolio Data component also acts as decoupling mechanism, allowing users to maintain and evolve all the other components of the ATDx tool independently.
Controller. It implements the coordination and communication mechanism of the ATDx tool. This component is responsible to set up and run the ATDx tool execution by interacting with the components ATDxCore, Analysis Tool Factory, and Report Generator Factory.

[5]For the sake of readability, these data transmission relations between the Portfolio Data and the ATDxCore, Analysis Tool Factory, and Report Generator Factory components are not shown in Figure 1.

2.2. Input and Output

The tool takes as input two configuration files and two additional files containing raw data.
Main Configuration. This configuration file defines which analysis tool(s) must be used and the location in the file system of the other files required by the the ATDx tool. An example configuration file is reported in Listing 1.
Report Generation Configuration. This configuration file is used to specify (i) the format of the report generated by the ATDx tool, (ii) if the reports should be stored on the Flask server, and (iii) the number of top source code entities containing AR violations to be included in the report (see also ATDx Reports).

Listing 1: Example of Main Configuration

```
{
    "tool": "SonarCloud",
    "rules_location": "triplets.json",
    "projects_location": "
        Apache_projects.json",
    "measures": "measures.json" ,
    "counted_issues": "ar_issues.json",
    "arch_issues": "arch_issues.json"
    "files_suffix": "new_"
}
```

Listing 2: Example of 3-tuple definition

```
"triple": {
    "java:S2975":{
        "granularity_level": "classes",
        "dimensions": [
            "inheritance",
            "jvms"
        ]
    }
}
```

ATDx Metadata. This file contains the metadata required for running an instance of the ATDx tool. Such file is composed of (i) the list of software projects to be analyzed and (ii) the 3-tuple definitions, consisting of AR name identifiers, and their associated granularity level and ATDDs. An example of 3-tuple definition is reported in Listing 2, where the java:S2975 rule is associated to the granularity level “classes” and two dimensions called “inheritance” and “jvms”.
ATDx Data. This file contains the raw data produced by previous executions of a source code analysis tool. This data is processed by the ATDxCore component to execute the actual portfolio analysis.

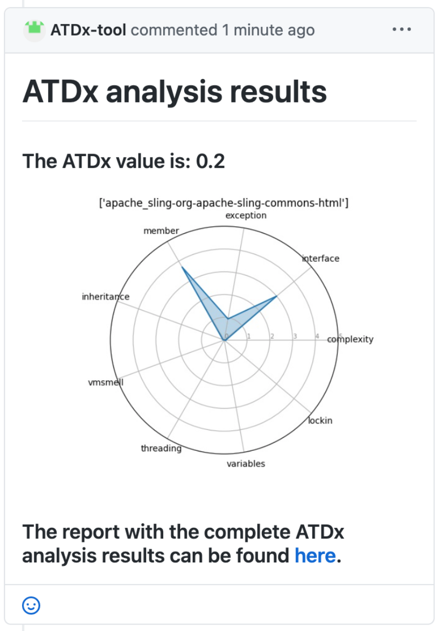
The ATDx tool produces two different types of output: ATDx reports and GitHub comments.
ATDx Reports. The ATDx analysis results are summarized in reports, which comprise a description of the ATDD, a radar chart depicting the ATDD values of projects, and a table with the top source code entities (*e.g.,* Java classes) containing AR violations, grouped by ATDD dimension. For traceability, the report also includes links to the raw data used in the analysis and the analyzed source code in the original repository. A sample ATDx report is made available online for completeness[6].
GitHub Comments. The GitHub comments produced by the ATDx tool includes the general ATDx index value calculated by the tool, a radar chart, and a link to the complete ATDx report (see Figure 2).

3. Implemented Plug-ins

In addition to the ATDx tool, we make available two already-implemented plug-ins, namely: SonarCloud Miner (an analysis tool) and Markdown Generator (a report generator). Such plug-ins allow to perform a complete execution of the ATDx tool out-of-the-box.

[6]https://tinyurl.com/ATDxSamleReport

SonarCloud Miner. Plug-in providing the capability to automatically gather AR violations to be used for the ATDx analysis from SonarCloud[7], a platform providing a dataset of pre-computed SonarQube analysis results of over 90K open-source software projects. The SonarCloud Miner plug-in can be run by specifying a set of software projects available on the platform.
Markdown Generator. Plug-in allowing users to automatically generate ATDx reports in Markdown, a popular markup language supporting conversion to HTML, which is currently utilized and endorsed in the technology stacks of some prominent software companies (*e.g.*, GitHub).



ATDx-tool commented 1 minute ago

ATDx analysis results

The ATDx value is: 0.2

The report with the complete ATDx analysis results can be found here.

Figure 2: Example of GitHub comment generated by the ATDx tool.

4. Conclusions and Future Work

With this paper we make available a Python-based implementation of ATDx [3], an approach leveraging software portfolio analysis to provide an overview of the ATD detected by third-party source code analysis tools. The tool follows a plugin-based architecture, allowing users to adapt the analysis tool and produced outputs according to their specific needs. In addition to the tool implementation, we provide two already-implemented plug-ins, allowing users to execute a complete run of the ATDx tool out-of-the-box. As future work, we plan to extend the tool by implementing more plug-ins, and apply the tool in the wild, by tuning the implementation of plug-ins according to the specific needs of practitioners in the context of large-scale industrial software development. In addition, we will develop a UI to intuitively configure the ATDx tool, instead of dealing with JSON configuration files. Finally, we intend to extend the ATDx tool with a history mechanism, allowing users to conduct longitudinal analyses on the evolution of ATD in their software systems over time.

References

[1] P. Kruchten, R. L. Nord, I. Ozkaya, Technical debt: From metaphor to theory and practice, IEEE Software 29 (2012) 18–21. doi:https://doi.org/10.1109/MS.2012.167.
[2] R. Verdecchia, I. Malavolta, P. Lago, Architectural technical debt identification: The research landscape, in: 2018 IEEE/ACM International Conference on Technical Debt (TechDebt), IEEE, 2018, pp. 11–20. doi:https://doi.org/10.1145/3194164.3194176.
[3] R. Verdecchia, P. Lago, I. Malavolta, I. Ozkaya, ATDx: Building an architectural technical debt index., in: ENASE, 2020, pp. 531–539. doi:https://doi.org/10.5220/0009577805310539.
[4] G. Erich, J. Vlissides, R. Helm, R. Johnson, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994, pp. 107–110. ISBN:0201633612.

[7]https://sonarcloud.io