vFunction

Application

Example Application

Measurement

Assessment Report

9/28/2022

Assessment Report

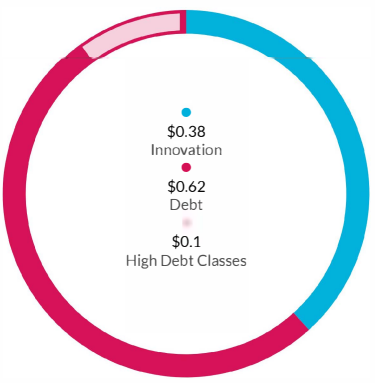
Cost of Innovation

The Assessment Report shows the level of technical debt and architectural complexity in your application. Through static analysis, the report exposes how the dependencies between the classes affect the modularity of the code, the risk of impacting parts of the application and the overall level of technical debt. This level of debt affects the rate and cost of innovation and is a key business measure for the application.

This chart shows how the level of technical debt affects the cost of innovation for the application. It tries to assess if one would spend $1 on this application, how much of it will actually be put to innovation, and how much to repay the technical debt within the application.

The classes that contribute the most to the overall debt, by adding complexity and risk throughout the application, are listed in page 2 of the report. The post-refactor calculation assesses the level of technical debt assuming these classes were refactored, and their level of debt reduced.

Pre-Refactor



61.7% - Technical debt

The report calculates the TCO of the app, compared to an app with no technical debt.

Pre-Refactor
Total cost of ownership multiplier
x2.6

Post-Refactor
Total cost of ownership multiplier
x2.1

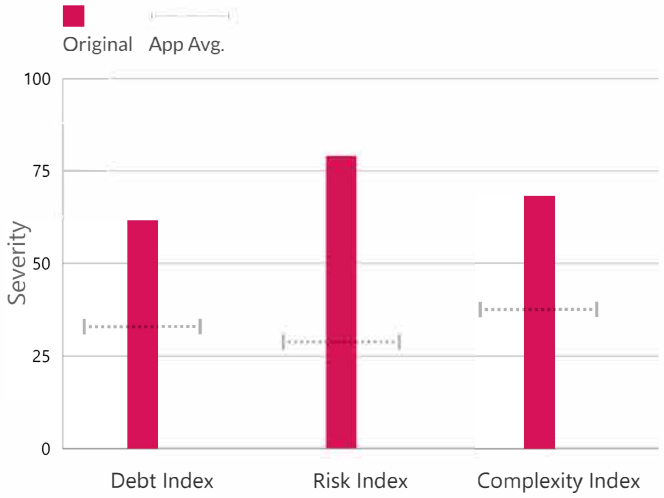
Tech Debt Metrics

Complexity is the degree to which class dependencies are entangled between themselves, reducing the level of modularity of the code.

Risk is correlated to the length of the dependencies. How likely is a change in the application to affect a seemingly unrelated part of it.

The debt is an overall calculation, taking into account complexity, risk and other factors to provide a single metric to prioritize the application for modernization.

The average of these three metrics in the portfolio of applications is presented to further help with the prioritization.



	Debt Index	Risk Index	Complexity Index
score	61.71	79.11	68.12
App Avg.	33	29	37
Distance from Avg.	+87%	+175%	+82%

Assessment Report

Aging Frameworks

An important part of technical debt is the aging frameworks that are used in the application. Aging frameworks are a source of security risk, and indicate an increased effort to upgrade the frameworks to their latest versions. The use of some aging frameworks will also block the application from being considered cloud-native.

This list below shows the names of the jars of the aging framework that are used in the application.

More details on the various frameworks is available in the online version of this report. This information includes the names of the actual jars, the current version and the newest version of the jar, and the maven coordinates of the framework's most recent version.

Each framework, based on its current and newest version receives a tag of either modern, aging, or unknown, in case the specific jar was not found in the vFunction database.

Your application is using

40%

Modern Framework

0%

Unknown Framework

60%

Aging Framework

Number of Classes

3638

Min Compile Version

1.0.2

vFunction Framework Classification

Modern Framework

A used framework was found to be of a recent version, up to the latest minor release.

Unknown Framework

A used framework was not identified by the vFunction platform. This may be a proprietary framework.

Aging Framework

A used framework was found to be older than the newest release by at least one minor version, and should be updated.

Top Debt Classes

Name

Aging Frameworks (showing 10 of 64)

Name	Version
example	1.4
example	1.4
example	8.1.0
example	7.6.0
example	1.9.4
example	1.8.3-kuali-4
example	3.2.2
example	4.4
example	62
example	3.1

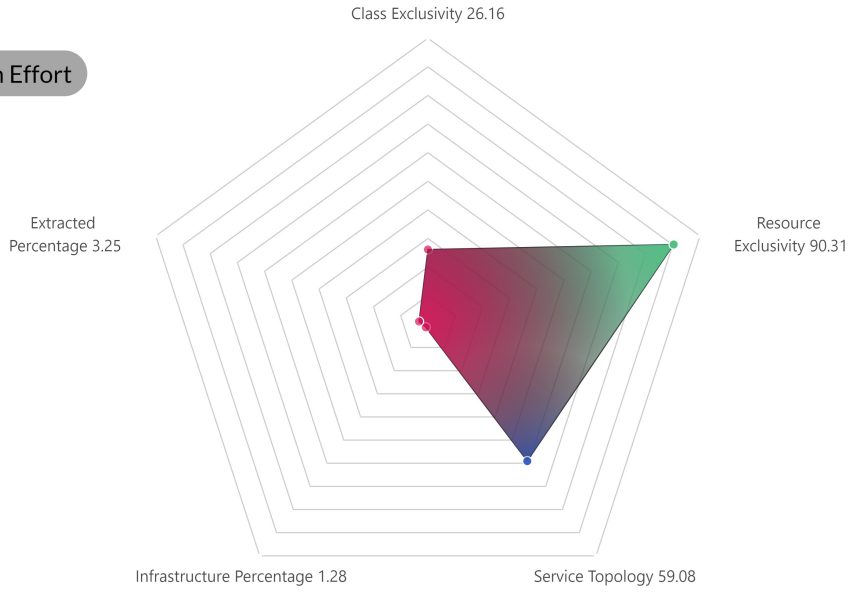
Refactoring Effort

Assessment Score 5 Very high Effort

The radar chart shows the **key modernization factors** of the application. The larger the area is covered by the graph, the more the application is suited for modernization.

While these parameters do not directly indicate the complexity of the application itself, they indicate the level of effort that is required to re-architect the application.

This assessment is based on a specific measurement, and depends on the analysis that was done on it.



The vFunction Refactoring Effort Score is ranked from 1 to 5. The higher the score, the higher the effort.

1	2	3	4	5
Low effort	Medium effort	Average effort	High effort	Very high effort
0%-19%	20%-39%	40%-59%	60%-79%	80%-100%

Extracted Percentage: Low

The percentage of classes not required in the monolith after extracting the services. To increase, add entry points closer to the root of the application.

Class Exclusivity: Low

The percentage of classes exclusive to the services. To increase, review the non-exclusive classes of the services.

Resource Exclusivity: High

The percentage of resources exclusive to the services. To increase, review the non-exclusive resources in the identified services.

Service Topology: Med

The number of service-to-service calls required to call any service. To increase, reduce the number of shared services.

Common Percentage: Med

The level of common classes found. To increase, review the common classes, mark business logic classes as non-common and mark infra jars.

Analysis Output

Services	14
Entry Points	103
Classes	390
Common Classes	701
Resource Exclusivity	96%
Class Exclusivity	77%
Extracted Percentage	38%

Identified Services (14)

AQEActionServlet.process() (+3)
AQEFacilityJDBCPersistentObje
AQELinkTag
AQEToAQAServiceSoapBindingS
CauseAreaLookupRunnable
CheckitemService
DefectCodesAndCategoriesLook
DisplayEventMiscLookupRunnab
EmailService
EventService
EventServiceSoapBindingSkeletc
...