CAST

Open Source Risk Scorecard

Priority insights and actions to reduce open source risks
due to security vulnerability, IP / licensing, and obsolescence exposures

Sample Report

Challenge

Over 70% of applications utilize open source components which introduces legal, security, and obsolescence risks.* However, traditional approaches to implementing Software Composition Analysis (SCA) are falling short:

- Slow and cumbersome rollout
- Increasingly complicated and expensive
- Compounding Developer “Alert Fatigue”

Ultimately, open source risks can get lost in the noise and ignored. But, there is a smarter approach to SCA…

CAST Highlight acts as an Open Source ‘command center’ across all applications, without disrupting developers.

*Gartner

Scope

This document is a sample of automatically generated SCA intelligence for a portfolio of 17 applications.

Key insights in this report include:

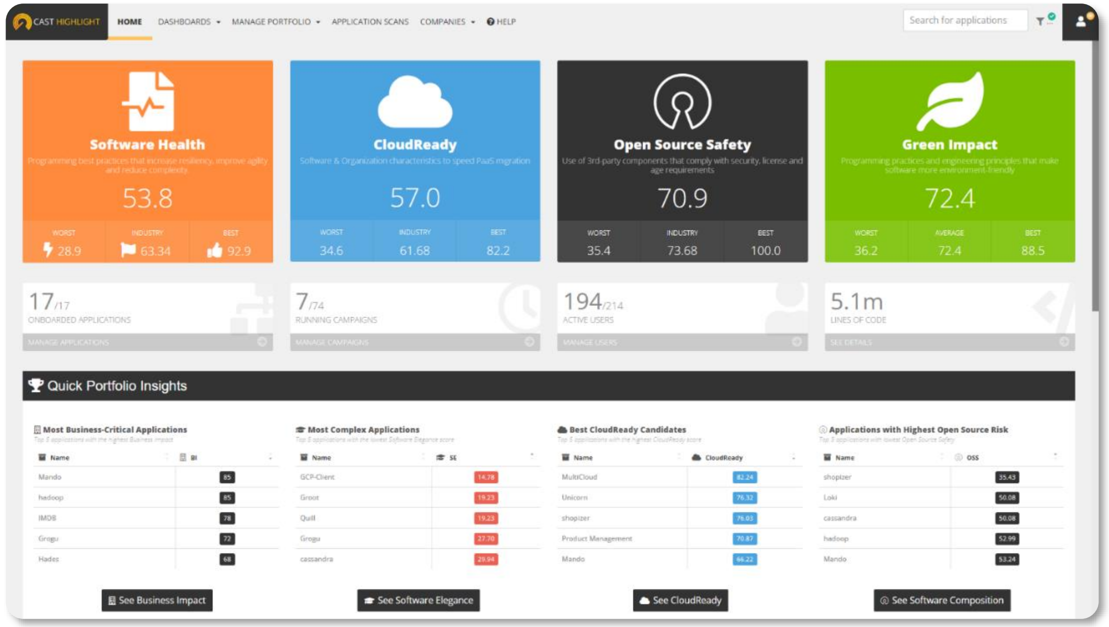
- Specific recommendations on how to reduce open source security, legal, and obsolescence risks
- Additional recommendations on how to:
  - optimize software maintenance costs, application resiliency, and tech debt
  - modernize each application to be cloud native
  - make software greener

CAST Highlight was used to produce the intelligence in a few hours by automatically understanding the source code and capturing qualitative information via a built-in survey capability.

See now

Request demo

Agenda



Software Health
Programming best practices that increase resiliency, improve agility and reduce complexity
53.8
WORST 28.9 | INDUSTRY 63.34 | BEST 92.9

CloudReady
Software & Organization characteristics to speed PaaS migration
57.0
WORST 34.6 | INDUSTRY 61.68 | BEST 82.2

Open Source Safety
Use of 3rd-party components that comply with security, license and age requirements
70.9
WORST 35.4 | INDUSTRY 73.68 | BEST 100.0

Green Impact
Programming practices and engineering principles that make software more environment-friendly
72.4
WORST 36.2 | AVERAGE 72.4 | BEST 88.5

17/17 ONBOARDED APPLICATIONS — MANAGE APPLICATIONS
7/74 RUNNING CAMPAIGNS — MANAGE CAMPAIGNS
194/214 ACTIVE USERS — MANAGE USERS
5.1m LINES OF CODE — SEE DETAILS

Quick Portfolio Insights

Most Business-Critical Applications
Top 5 applications with the highest Business Impact

Name	BI
Mando	85
hadoop	85
IMDB	78
Grogu	72
Hades	68

See Business Impact

Most Complex Applications
Top 5 applications with the lowest Software Elegance score

Name	SE
GCP-Client	14.78
Groot	19.23
Quill	19.23
Grogu	27.70
cassandra	29.94

See Software Elegance

Best CloudReady Candidates
Top 5 applications with the highest CloudReady score

Name	CloudReady
MultiCloud	82.24
Unicorn	76.32
shopizer	76.03
Product Management	70.87
Mando	66.22

See CloudReady

Applications with Highest Open Source Risk
Top 5 applications with lowest Open Source Safety

Name	OSS
shopizer	35.43
Loki	50.08
cassandra	50.08
hadoop	52.99
Mando	53.24

See Software Composition

Portfolio snapshot

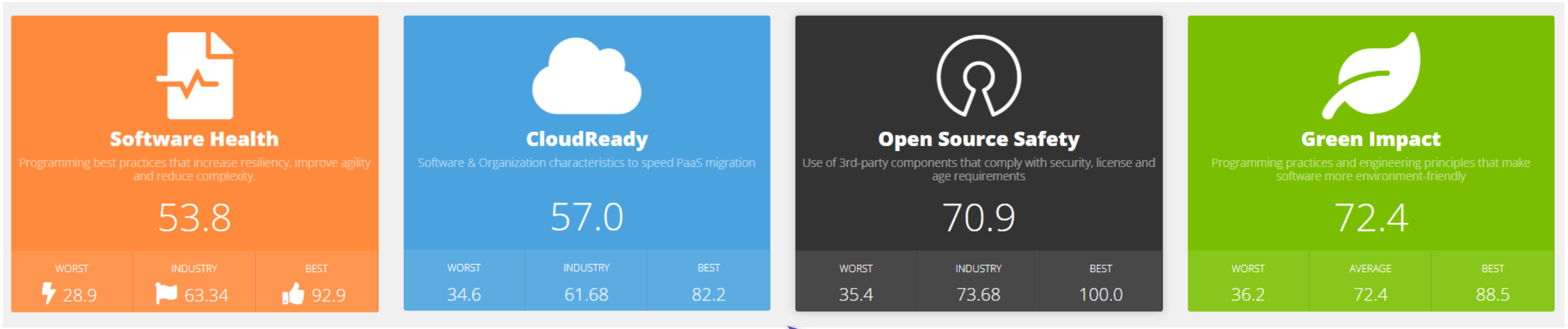
Application portfolio snapshot

17
applications

10
technologies
(programming languages)

5.1m
lines of code

527
open-source components



Software Health
Programming best practices that increase resiliency, improve agility and reduce complexity.
53.8

WORST	INDUSTRY	BEST
28.9	63.34	92.9

CloudReady
Software & Organization characteristics to speed PaaS migration
57.0

WORST	INDUSTRY	BEST
34.6	61.68	82.2

Open Source Safety
Use of 3rd-party components that comply with security, license and age requirements
70.9

WORST	INDUSTRY	BEST
35.4	73.68	100.0

Green Impact
Programming practices and engineering principles that make software more environment-friendly
72.4

WORST	AVERAGE	BEST
36.2	72.4	88.5

A portfolio snapshot provides a summary of the portfolio and top line metrics for all applications. (All metrics are defined in the appendix.)

CAST

Application portfolio snapshot

Technology	Size (LOC)	Resiliency	Agility	Elegance
Java	2.5M	🟥 53.44	🟨 58.94	🟥 36.88
C#	1.6M	🟨 78.91	🟨 57.41	🟨 61.97
Cobol	712K	🟥 45.10	🟨 56.00	🟥 37.86
VB	202K	🟥 54.15	🟨 63.27	🟨 49.52
C/C++	104K	🟨 68.18	🟨 66.08	🟨 44.90
Javascript	26k	🟨 66.66	🟥 53.90	🟩 73.11
Python	20k	🟥 61.26	🟨 63.89	🟨 56.18
Ksh	11k	🟨 67.74	🟨 66.71	🟩 88.37
JSP	6k	🟥 52.93	🟨 68.74	🟩 98.15
T/SQL	1k	🟩 90.00	🟥 49.36	🟩 77.15

The portfolio snapshot also includes the portfolio demographics broken down by technology and health scores (resiliency, agility, elegance).

Software Composition Analysis

Software Composition Analysis Section

This section of the report contains key insights generated by CAST Highlight on the Software Composition (open source risks) of applications that should addressed and monitored regularly including:

- Security vulnerabilities to be addressed
- Risky open source licenses that create potential legal exposures
- Summarized action plan for the application portfolio

Software Composition Analysis Overview

Check Third-Party Vulnerabilities

Open source is one of the major entry points for hackers. It is critical to identify if the third-party components in use contain security vulnerabilities.

Control Open Source License Compliance

Open source licensing can be complex and confusing. Visibility on the licenses used by open source components is required to detect any restrictive license compliance issues.

Reduce Technology Obsolescence

Open source components can become out of date or unsupported resulting in operational risks and outages. These out of date components must be detected and replaced with supported components.

CAST

Security Vulnerabilities Overview

Third-Party Component Vulnerabilities

Portfolio Insights & Top 5

88	159	169	13	23
CRITICAL	HIGH	MEDIUM	LOW	ADVISORY

Top 5	Business Impact	Possible Vulnerabilities
hadoop	85.3	⬛ 9 🟥 11 🟨 34 ◼ 2 □ 6
Grogu	71.6	⬛ 1 🟥 0 🟨 2 ◼ 0 0
Hades	68.3	⬛ 0 🟥 10 🟨 3 ◼ 0 □ 3
GCP-Client	56.9	⬛ 4 🟥 8 🟨 17 ◼ 1 4
Loki	49.5	⬛ 29 🟥 50 🟨 26 ◼ 3 □ 1

The number and criticality of open source security vulnerabilities are identified across the portfolio.

Security Vulnerabilities Detail

Vulnerabilities

Application	Components
Hadoop	cxf-rt-transports-http-jetty 3.0.3, slf4j-api 1.7.7, jsch 0.1.42
Grogu	Microsoft.Practices.EnterpriseLibrary.Logging 4.1.0.0, Microsoft.Practices.EnterpriseLibrary.Common 4.1.0.0
Hades	cxf-rt-frontend-jaxws 2.7.5
GCP-Client	minimatch 3.0.0, useragent 2.1.12, qs 2.3.3, decamelize 1.1.1, parsejson 0.0.3, hapi 15.x.x,
Loki	tomcat-embed-core 7.0.73, slf4j-api 1.7.7, cxf-rt-frontend-jaxws 2.7.12, is-my-json-valid 2.12.0, ua-parser-js 0.7.12, marked 0.3.6, minimatch 3.0.0, useragent 2.1.11, jquery 1.7.2, hibernate-validator 4.2.0.Final,
Other applications	openjpa-persistence-jdbc 2.1.1, commons-fileupload 1.2.1, jackson-databind 2.5.3, dom4j 1.6.1, jsoup 1.8.1, derby 10.1.1.0 ...

Specific open-source components with vulnerabilities in each application are identified.

License Risk Overview

Third-Party Component License Risk
Portfolio Insights & Top 5

531 COMPONENTS	25 HIGH RISK	50 MEDIUM RISK	412 LOW RISK	12 UNDEFINED

Top 5	Business Impact	Licenses
hadoop	85.3	🟥 17 🟨 16 🟩 181 ⬜ 2
Mando	85.3	🟥 1 🟨 0 🟩 0 ⬜ 1
Grogu	71.6	🟥 0 🟨 0 🟩 2 ⬜ 0
Hades	68.3	🟥 1 🟨 16 🟩 106 ⬜ 4
roslyn	63.2	🟥 0 🟨 0 🟩 2 ⬜ 0

The number and risk levels of open source licenses are identified across the portfolio.

CAST

License Risk Detail

License Risk

Application	3rd-Party Components	Licenses
Hadoop	7	MIT License (2), Apache 2.0 (1), BSD-3 New
Mando	12	Apache 2.0 (3), GNU Affero GPL 3.0 (2)
Grogu	4	MIT License (2), ISC License (1)
Hades	379	MIT License (358), ISC License (39), Apache 2.0 (16), Eclipse 2.0 (1), BSD 2 (14), GNU Affero GPL 3 (1), BSD 3 (1)
Roslyn	32	MIT License (2), Apache 2.0 (1), GNU GPL 3 (4)

Applications that use open source components with risky licenses are highlighted.

Software Composition Recommendations

Hadoop: Upgrade jsh component to latest version to reduce critical vulnerability risk

Hades:

- Upgrade hibernate component to latest version to reduce critical vulnerability risk
- Replace component that uses the GNU GPL license to avoid copyleft licensing risk

Mando: Replace component that uses the GNU GPL license to avoid copyleft licensing risk

Roslyn: Replace component that uses the GNU GPL license to avoid copyleft licensing risk

Additional recommendations:

- Continuously monitor Health of each application to understand opportunities to improve resiliency and agility.
- Analyze Cloud Maturity of each application to modernize the portfolio.
- Investigate Green Impact of each application to identify opportunities for reducing energy consumption and carbon emissions.

Specific recommendations on how to reduce open source vulnerability and license risk are summarized.

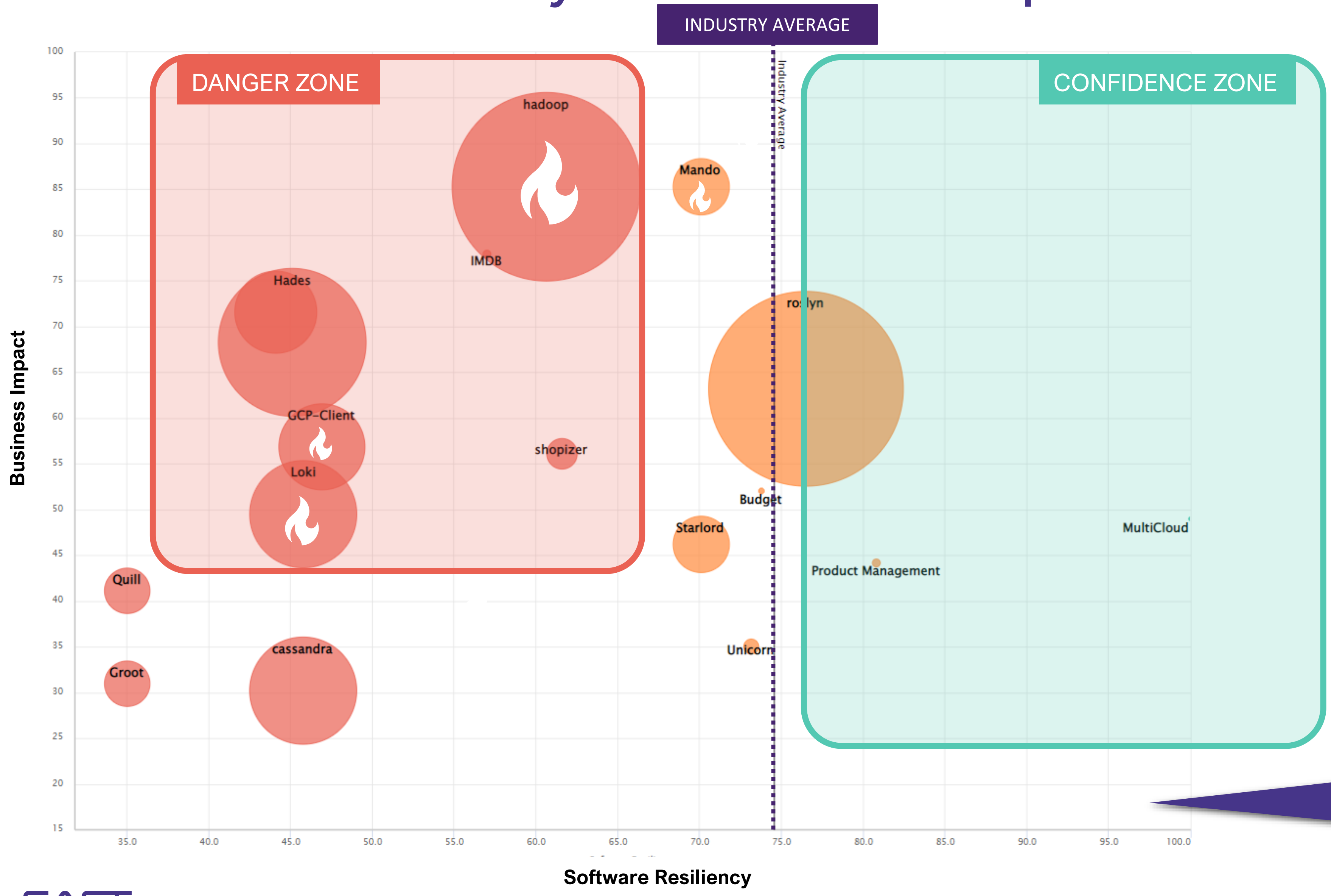
Software Health

Software Health section

This section of the report contains key insights generated by CAST Highlight on the Software Health of applications that should be continuously optimized including:

- Applications that are business critical and have low Resiliency
- Specific improvement opportunities within the code to improve Resiliency
- Applications where software maintenance costs and resource allocation can be optimized
- Summarized action plan for the application portfolio

Software Resiliency vs Business Impact



DANGER ZONE

Hadoop	Loki
Hades	IMDB
Grogu	Shopizer
GCP-Client	

CONFIDENCE ZONE

MultiCloud	Product Management
Roslyn	

VULNERABILITIES

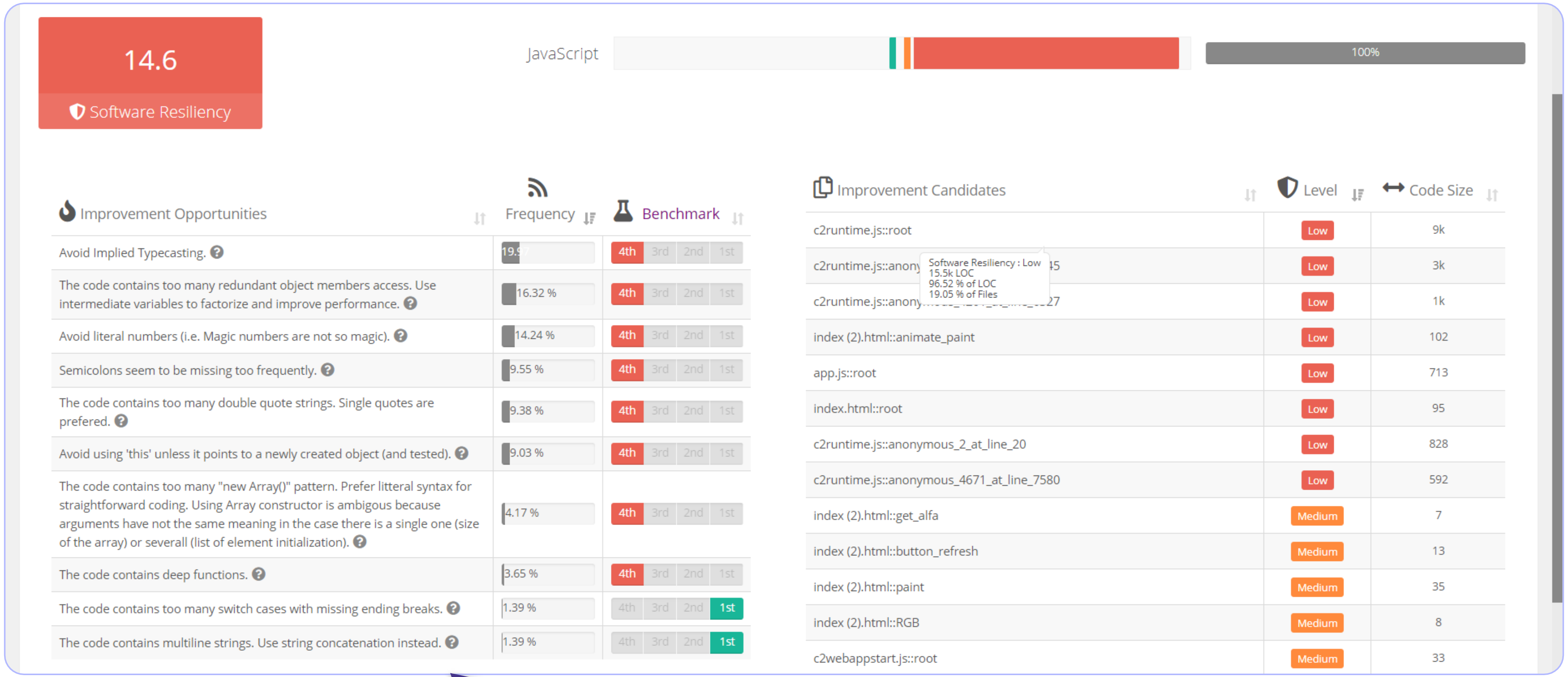
Loki	Mando
Hadoop	GCP-Client

Detected Common Vulnerabilities from Software Composition Analysis

Application Health insights (such as Resiliency) are analyzed to ensure unhealthy applications are improved

CAST

Software Resiliency for Loki Application



14.6
Software Resiliency

JavaScript 100%

Improvement Opportunities	Frequency	Benchmark
Avoid Implied Typecasting.	19.97	**4th** 3rd 2nd 1st
The code contains too many redundant object members access. Use intermediate variables to factorize and improve performance.	16.32 %	**4th** 3rd 2nd 1st
Avoid literal numbers (i.e. Magic numbers are not so magic).	14.24 %	**4th** 3rd 2nd 1st
Semicolons seem to be missing too frequently.	9.55 %	**4th** 3rd 2nd 1st
The code contains too many double quote strings. Single quotes are prefered.	9.38 %	**4th** 3rd 2nd 1st
Avoid using 'this' unless it points to a newly created object (and tested).	9.03 %	**4th** 3rd 2nd 1st
The code contains too many "new Array()" pattern. Prefer litteral syntax for straightforward coding. Using Array constructor is ambigous because arguments have not the same meaning in the case there is a single one (size of the array) or severall (list of element initialization).	4.17 %	**4th** 3rd 2nd 1st
The code contains deep functions.	3.65 %	**4th** 3rd 2nd 1st
The code contains too many switch cases with missing ending breaks.	1.39 %	4th 3rd 2nd **1st**
The code contains multiline strings. Use string concatenation instead.	1.39 %	4th 3rd 2nd **1st**

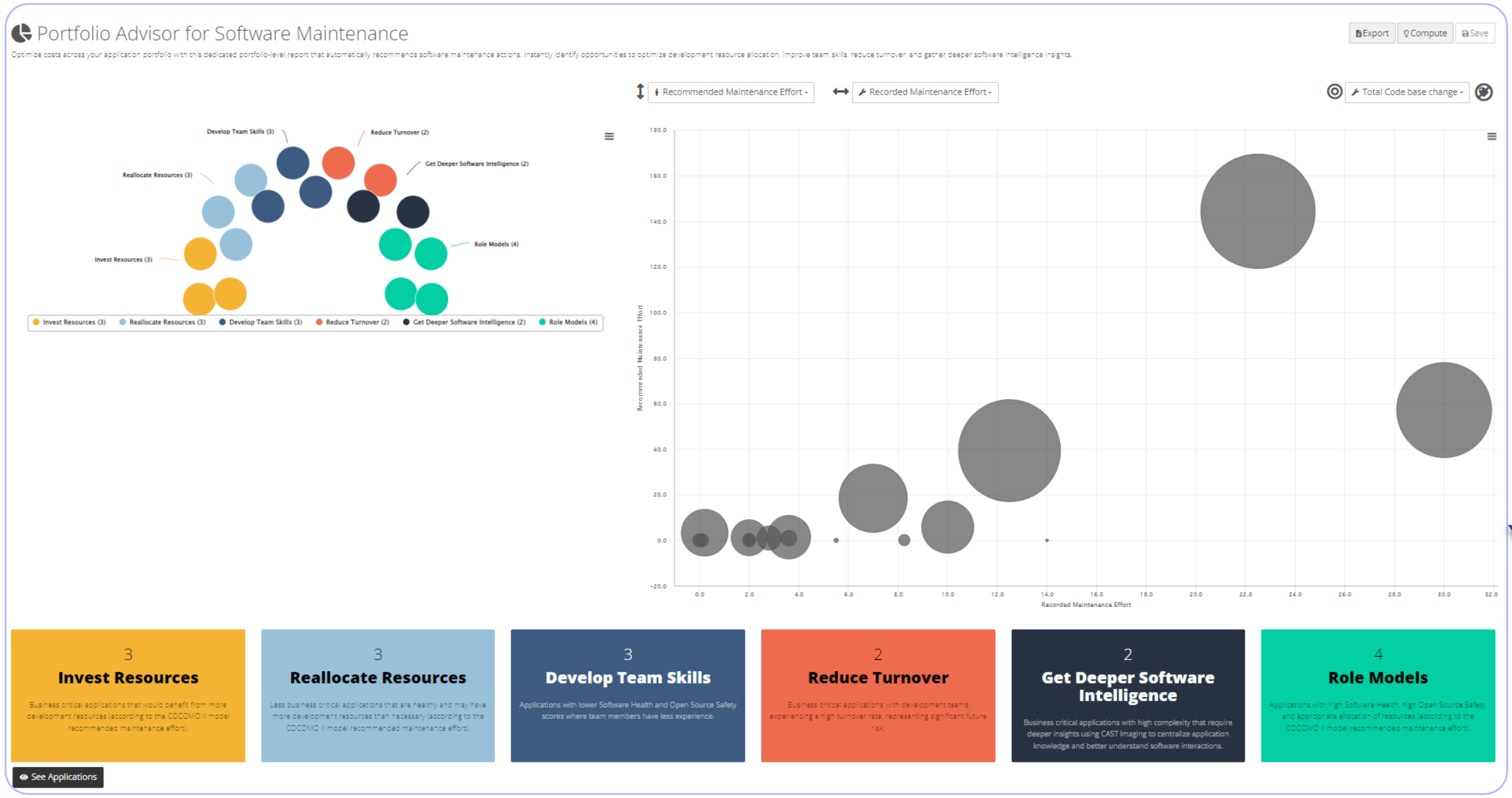
Improvement Candidates	Level	Code Size
c2runtime.js::root	Low	9k
c2runtime.js::anony 45	Low	3k
c2runtime.js::anony 27	Low	1k
index (2).html::animate_paint	Low	102
app.js::root	Low	713
index.html::root	Low	95
c2runtime.js::anonymous_2_at_line_20	Low	828
c2runtime.js::anonymous_4671_at_line_7580	Low	592
index (2).html::get_alfa	Medium	7
index (2).html::button_refresh	Medium	13
index (2).html::paint	Medium	35
index (2).html::RGB	Medium	8
c2webappstart.js::root	Medium	33

Software Resiliency : Low
15.5k LOC
96.52 % of LOC
19.05 % of Files

Unhealthy applications are analyzed at a deeper level to understand specific code-level improvement opportunities.

CAST

Portfolio Advisor for Software Maintenance


Export
Compute
Save
Recommended Maintenance Effort
Recorded Maintenance Effort
Total Code base change
Develop Team Skills (3)
Reduce Turnover (2)
Get Deeper Software Intelligence (2)
Reallocate Resources (3)
Role Models (4)
Invest Resources (3)
● Invest Resources (3) ● Reallocate Resources (3) ● Develop Team Skills (3) ● Reduce Turnover (2) ● Get Deeper Software Intelligence (2) ● Role Models (4)
Recommended Maintenance Effort
Recorded Maintenance Effort
See Applications

Portfolio Advisor for Software Maintenance

Optimize costs across your application portfolio with this dedicated portfolio-level report that automatically recommends software maintenance actions. Instantly identify opportunities to optimize development resource allocation, improve team skills, reduce turnover, and gather deeper software intelligence insights.

3
Invest Resources
Business critical applications that would benefit from more development resources (according to the COCOMO II model recommended maintenance effort).

3
Reallocate Resources
Less business critical applications that are healthy and may have more development resources than necessary (according to the COCOMO II model recommended maintenance effort).

3
Develop Team Skills
Applications with lower Software Health and Open Source Safety scores where team members have less experience.

2
Reduce Turnover
Business critical applications with development teams experiencing a high turnover rate, representing significant future risk.

2
Get Deeper Software Intelligence
Business critical applications with high complexity that require deeper insights using CAST Imaging to centralize application knowledge and better understand software interactions.

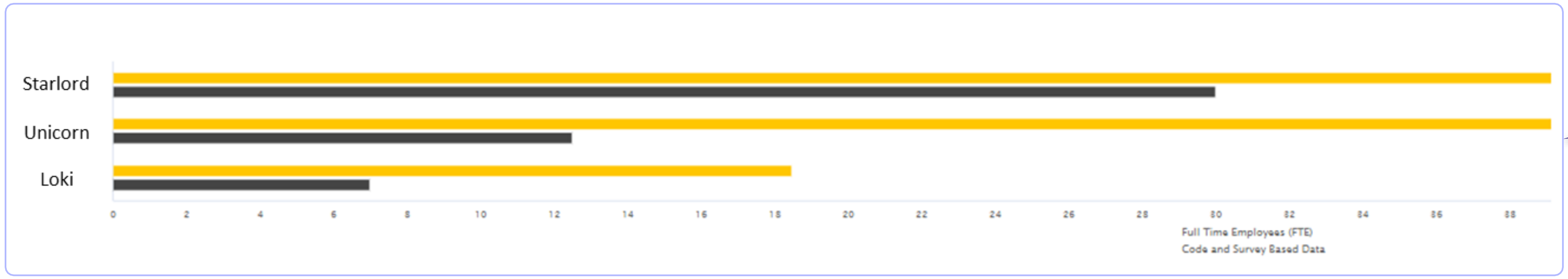
4
Role Models
Applications with high Software Health, high Open Source Safety, and appropriate allocation of resources (according to the COCOMO II model recommended maintenance effort).

The Portfolio Advisor for Software Maintenance automatically recommends actions to take on specific applications to optimize software maintenance costs and efficiency such as applications with too many or too few resources. It also identifies opportunities to develop team skills and reduce turnover.

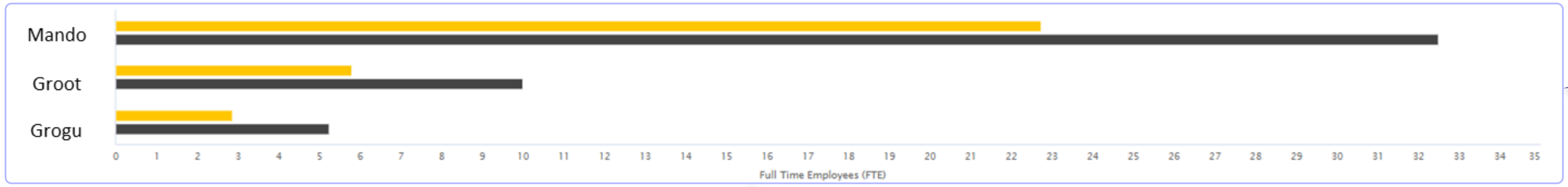
Software Maintenance Optimization

● Recorded Maintenance Effort

● Recommended Maintenance Effort



Likely too few resources on these applications.



Likely too many resources on these applications.

Software maintenance effort recommendations are based on comparing the *actual (recorded)* maintenance effort with the *recommended* maintenance effort (calculated automatically based on the COCOMO II industry standard model).

CAST

Software Health | Recommendations

Some applications have Resiliency scores that are severely low. Code alerts should be remediated to improve performance and reduce production outage risk:

- Hades
- Loki
- Grogu

Security Vulnerabilities were identified in a few applications and a deeper Software Composition Analysis should be performed to investigate the open source components in these applications further:

- Loki
- Hadoop
- Mando
- GCP-Client

Software Maintenance costs and efficiency can be optimized with the following actions:

- Invest resources – Starlord, Unicorn, Loki
- Reallocate resources – Mando, Groot, Grogu
- Develop team skills – Quill, Hadoop, Hades
- Reduce Turnover – GCP-Client, MultiCloud

Additional recommendations on: how to improve Software Health issues, potential security vulnerabilities to investigate, and software maintenance optimization actions are summarized.

CAST

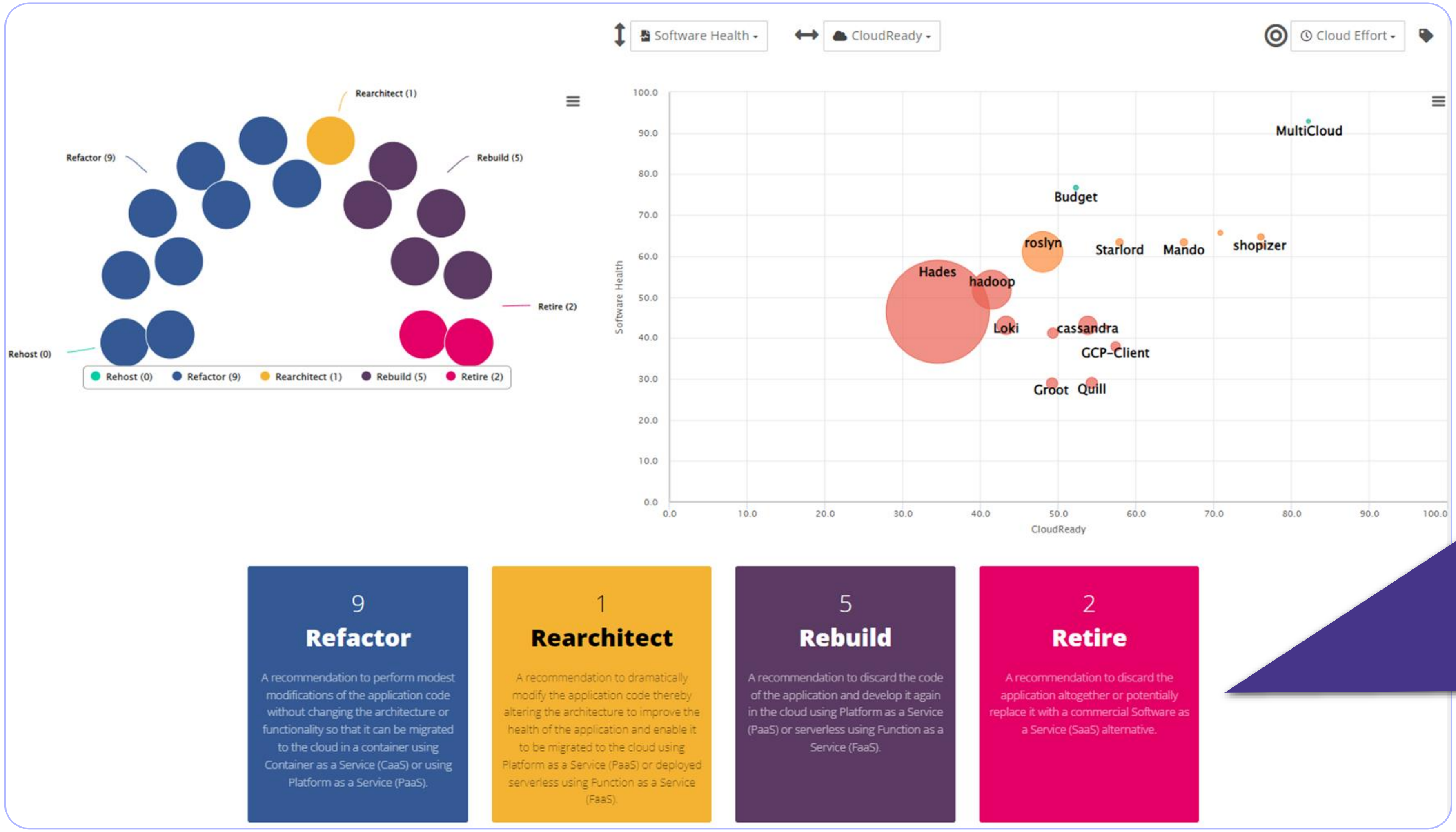
Cloud Maturity

Cloud Maturity Section

This section of the report contains key insights generated by CAST Highlight on the Cloud Maturity of applications including:

- Recommended modernization approaches for each application (Refactor, Rearchitect, Rebuild)
- Blockers to PaaS deployment, estimated effort to remove them, and the required code changes
- Recommended cloud native services that applications can adopt when deployed in a PaaS environment
- Summarized action plan for the application portfolio

Portfolio Advisor for Cloud



9
Refactor
A recommendation to perform modest modifications of the application code without changing the architecture or functionality so that it can be migrated to the cloud in a container using Container as a Service (CaaS) or using Platform as a Service (PaaS).

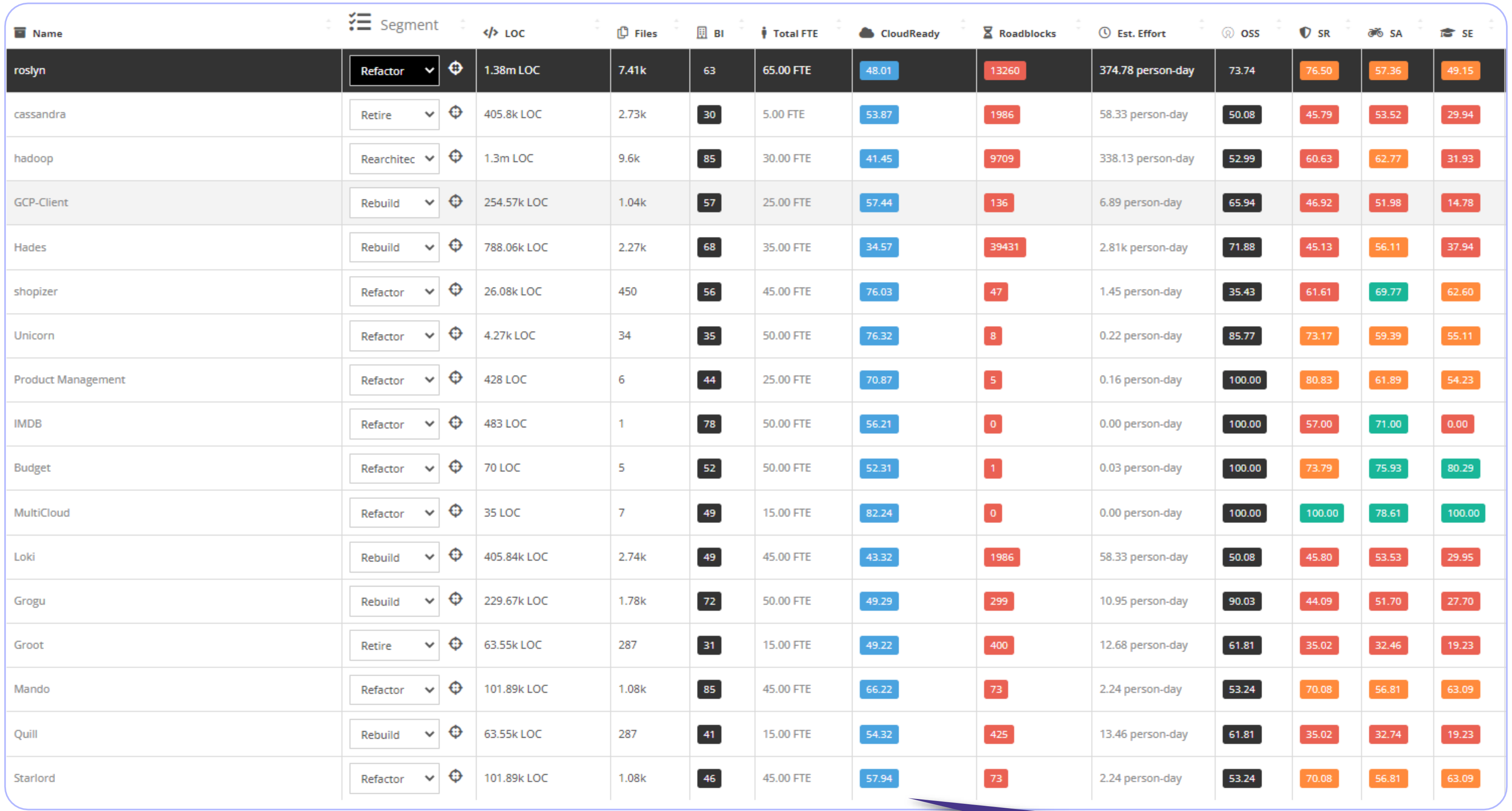
1
Rearchitect
A recommendation to dramatically modify the application code thereby altering the architecture to improve the health of the application and enable it to be migrated to the cloud using Platform as a Service (PaaS) or deployed serverless using Function as a Service (FaaS).

5
Rebuild
A recommendation to discard the code of the application and develop it again in the cloud using Platform as a Service (PaaS) or serverless using Function as a Service (FaaS).

2
Retire
A recommendation to discard the application altogether or potentially replace it with a commercial Software as a Service (SaaS) alternative.

The Portfolio Advisor for Cloud automatically segments each application and recommends the ideal modernization approach based on fact-based technical characteristics (via automated source code analysis) and qualitative criteria such as business impact (captured via survey).

Portfolio Advisor for Cloud

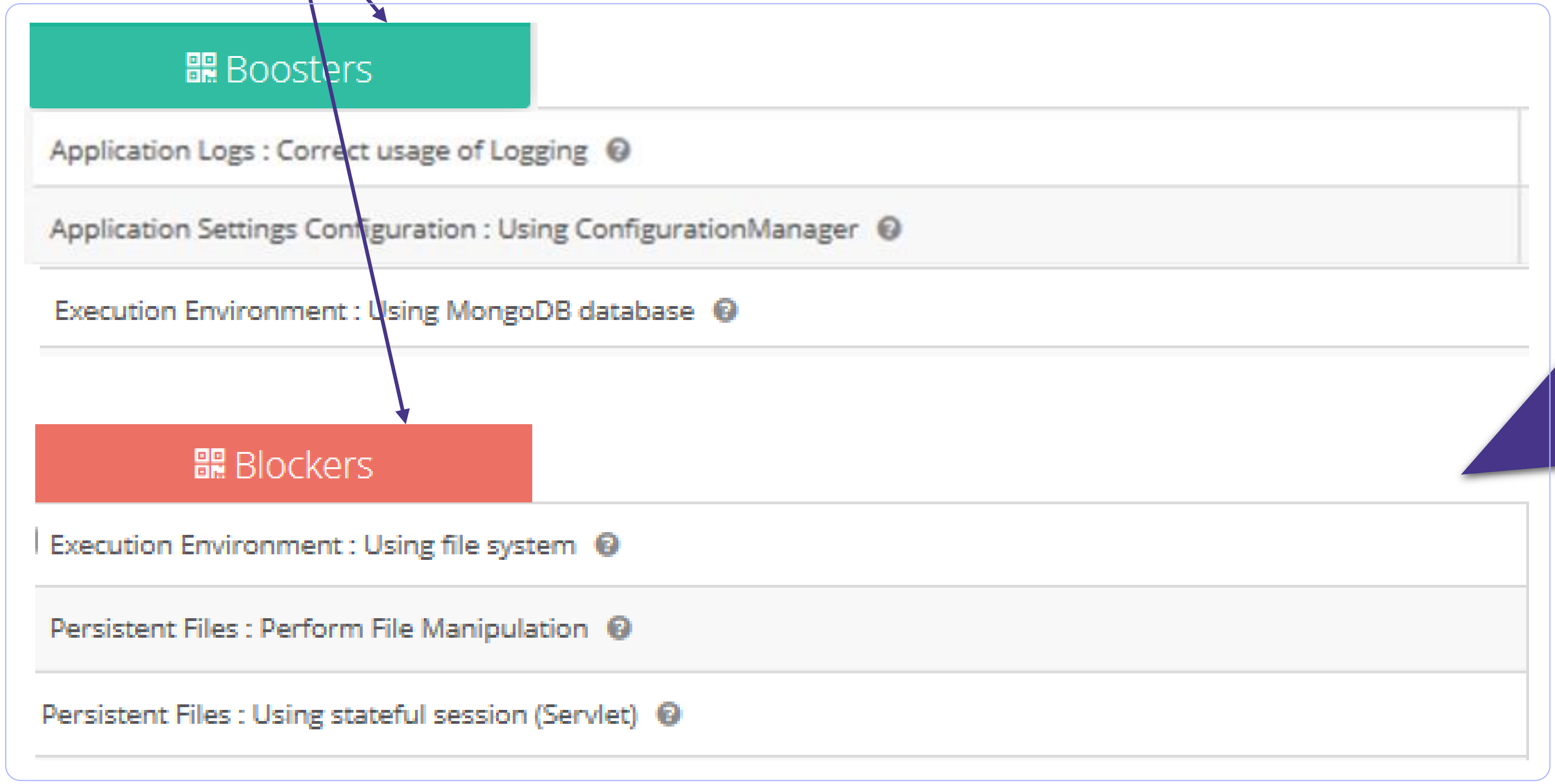


Name	Segment	LOC	Files	BI	Total FTE	CloudReady	Roadblocks	Est. Effort	OSS	SR	SA	SE
roslyn	Refactor	1.38m LOC	7.41k	63	65.00 FTE	48.01	13260	374.78 person-day	73.74	76.50	57.36	49.15
cassandra	Retire	405.8k LOC	2.73k	30	5.00 FTE	53.87	1986	58.33 person-day	50.08	45.79	53.52	29.94
hadoop	Rearchitec	1.3m LOC	9.6k	85	30.00 FTE	41.45	9709	338.13 person-day	52.99	60.63	62.77	31.93
GCP-Client	Rebuild	254.57k LOC	1.04k	57	25.00 FTE	57.44	136	6.89 person-day	65.94	46.92	51.98	14.78
Hades	Rebuild	788.06k LOC	2.27k	68	35.00 FTE	34.57	39431	2.81k person-day	71.88	45.13	56.11	37.94
shopizer	Refactor	26.08k LOC	450	56	45.00 FTE	76.03	47	1.45 person-day	35.43	61.61	69.77	62.60
Unicorn	Refactor	4.27k LOC	34	35	50.00 FTE	76.32	8	0.22 person-day	85.77	73.17	59.39	55.11
Product Management	Refactor	428 LOC	6	44	25.00 FTE	70.87	5	0.16 person-day	100.00	80.83	61.89	54.23
IMDB	Refactor	483 LOC	1	78	50.00 FTE	56.21	0	0.00 person-day	100.00	57.00	71.00	0.00
Budget	Refactor	70 LOC	5	52	50.00 FTE	52.31	1	0.03 person-day	100.00	73.79	75.93	80.29
MultiCloud	Refactor	35 LOC	7	49	15.00 FTE	82.24	0	0.00 person-day	100.00	100.00	78.61	100.00
Loki	Rebuild	405.84k LOC	2.74k	49	45.00 FTE	43.32	1986	58.33 person-day	50.08	45.80	53.53	29.95
Grogu	Rebuild	229.67k LOC	1.78k	72	50.00 FTE	49.29	299	10.95 person-day	90.03	44.09	51.70	27.70
Groot	Retire	63.55k LOC	287	31	15.00 FTE	49.22	400	12.68 person-day	61.81	35.02	32.46	19.23
Mando	Refactor	101.89k LOC	1.08k	85	45.00 FTE	66.22	73	2.24 person-day	53.24	70.08	56.81	63.09
Quill	Rebuild	63.55k LOC	287	41	15.00 FTE	54.32	425	13.46 person-day	61.81	35.02	32.74	19.23
Starlord	Refactor	101.89k LOC	1.08k	46	45.00 FTE	57.94	73	2.24 person-day	53.24	70.08	56.81	63.09

Additional statistics are provided for each application to further refine the roadmap.

Top Blockers & Boosters

Below are the top three Boosters and Blockers to cloud native found across the portfolio.



Application Logs : Correct usage of Logging

Application Settings Configuration : Using ConfigurationManager

Execution Environment : Using MongoDB database

Execution Environment : Using file system

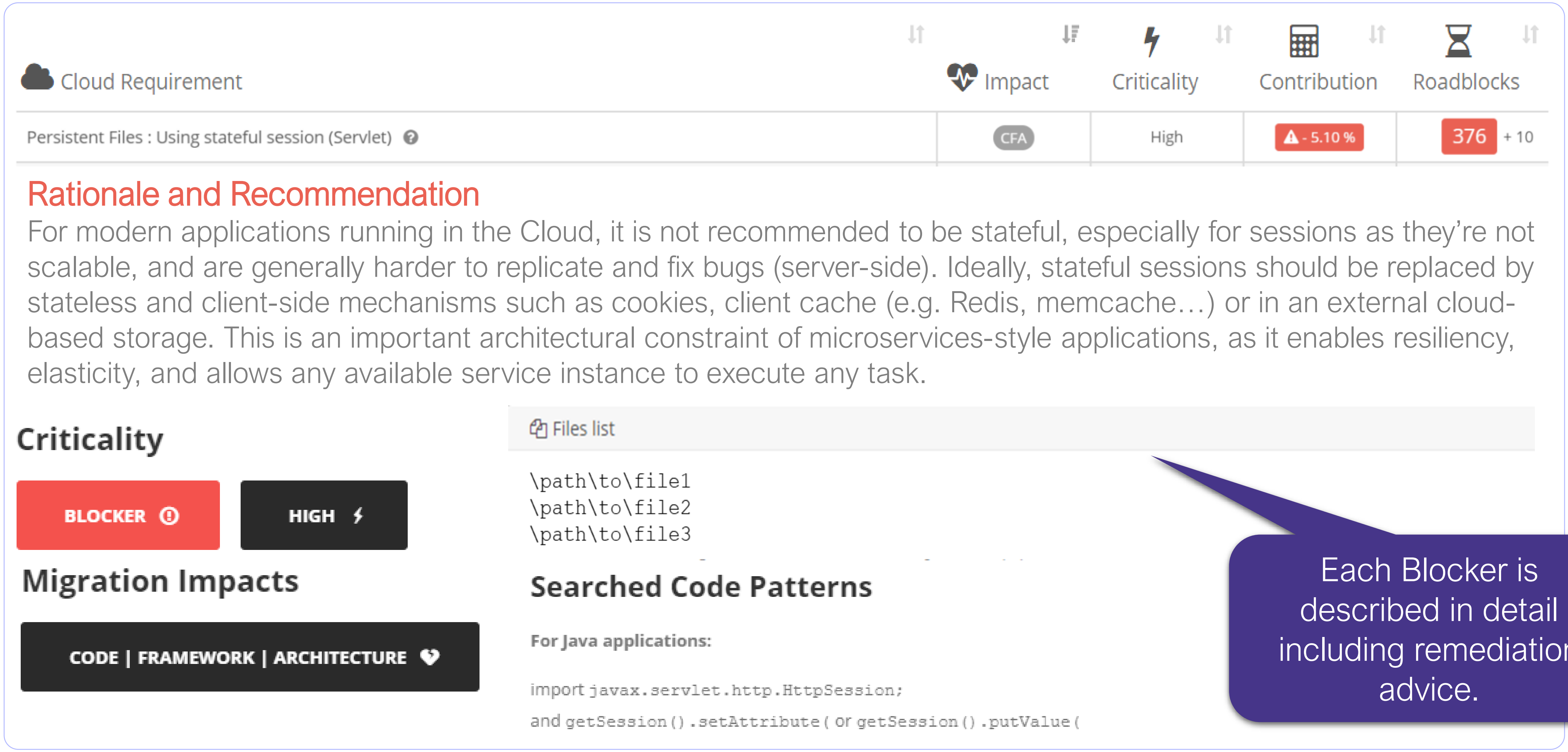
Persistent Files : Perform File Manipulation

Persistent Files : Using stateful session (Servlet)

Here are the top three PaaS Blockers and Boosters observed across the entire portfolio.

Blockers are code level issues that need to be addressed before the application can adopt cloud native services. These are described in more detail on the following pages.

Blocker Detail: Using Stateful Sessions



Cloud Requirement	Impact	Criticality	Contribution	Roadblocks
Persistent Files : Using stateful session (Servlet)	CFA	High	- 5.10 %	376 + 10

Rationale and Recommendation

For modern applications running in the Cloud, it is not recommended to be stateful, especially for sessions as they're not scalable, and are generally harder to replicate and fix bugs (server-side). Ideally, stateful sessions should be replaced by stateless and client-side mechanisms such as cookies, client cache (e.g. Redis, memcache…) or in an external cloud-based storage. This is an important architectural constraint of microservices-style applications, as it enables resiliency, elasticity, and allows any available service instance to execute any task.

Criticality

BLOCKER   HIGH

Migration Impacts

CODE | FRAMEWORK | ARCHITECTURE

Files list

```
\path\to\file1
\path\to\file2
\path\to\file3
```

Searched Code Patterns

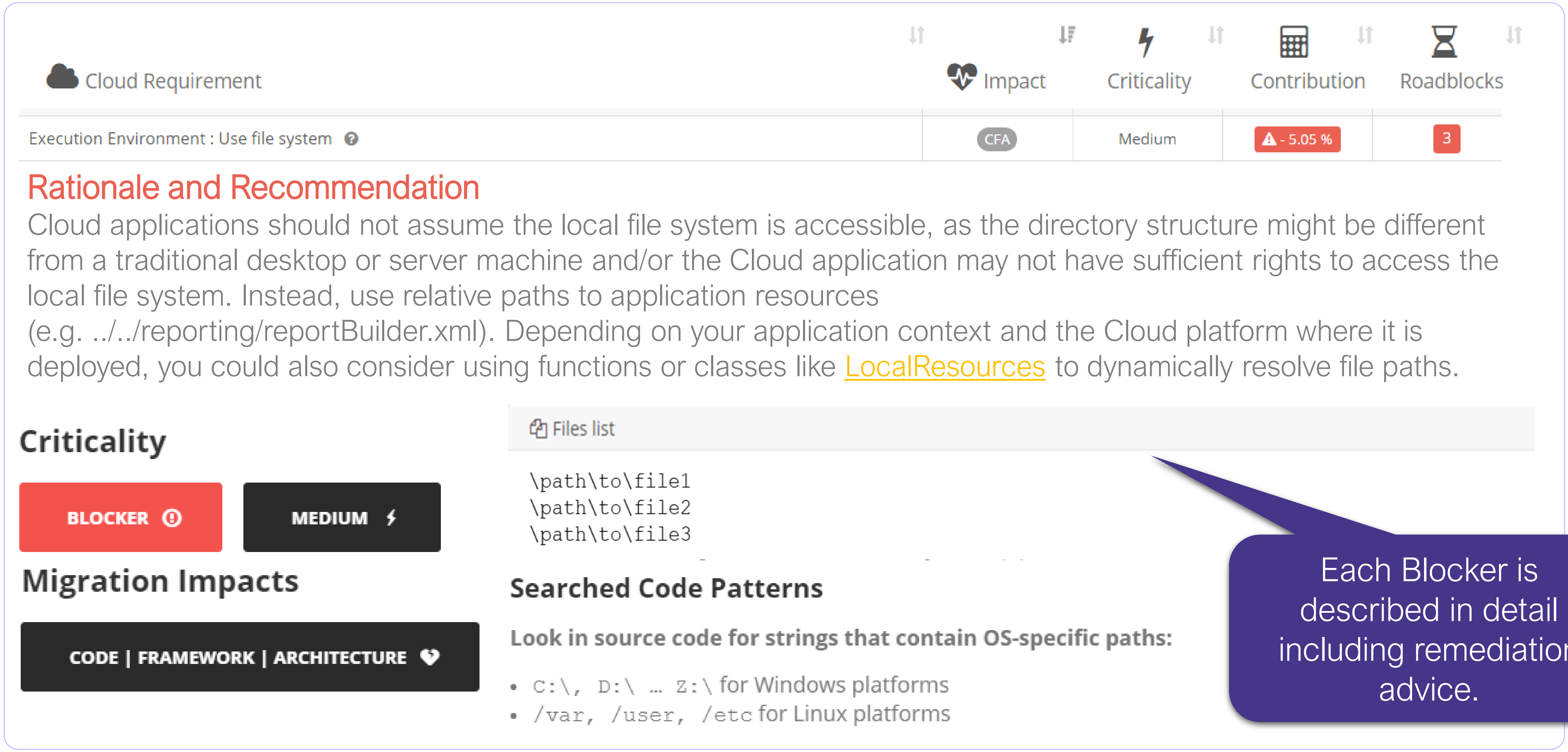
For Java applications:

import `javax.servlet.http.HttpSession;`

and `getSession().setAttribute(` or `getSession().putValue(`

Each Blocker is described in detail including remediation advice.

Blocker Detail: Use of File System



Cloud Requirement	Impact	Criticality	Contribution	Roadblocks
Execution Environment : Use file system	CFA	Medium	⚠ - 5.05 %	3

Rationale and Recommendation

Cloud applications should not assume the local file system is accessible, as the directory structure might be different from a traditional desktop or server machine and/or the Cloud application may not have sufficient rights to access the local file system. Instead, use relative paths to application resources (e.g. ../../reporting/reportBuilder.xml). Depending on your application context and the Cloud platform where it is deployed, you could also consider using functions or classes like LocalResources to dynamically resolve file paths.

Criticality

BLOCKER MEDIUM

Migration Impacts

CODE | FRAMEWORK | ARCHITECTURE

Files list

```
\path\to\file1
\path\to\file2
\path\to\file3
```

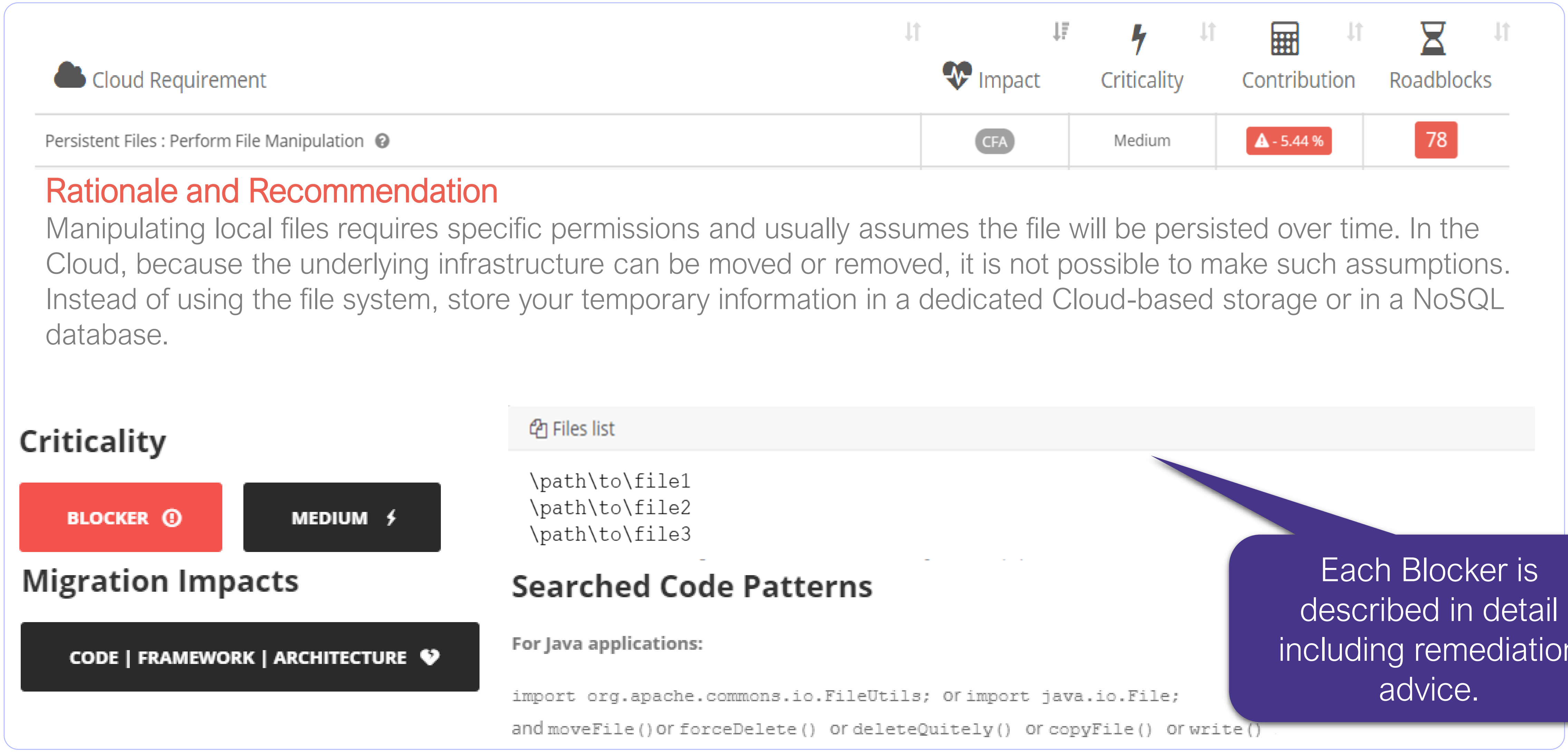
Searched Code Patterns

Look in source code for strings that contain OS-specific paths:

- `C:\, D:\ … Z:\` for Windows platforms
- `/var, /user, /etc` for Linux platforms

Each Blocker is described in detail including remediation advice.

Blocker Detail: Perform File Manipulation



Cloud Requirement	Impact	Criticality	Contribution	Roadblocks
Persistent Files : Perform File Manipulation	CFA	Medium	⚠ - 5.44 %	78

Rationale and Recommendation

Manipulating local files requires specific permissions and usually assumes the file will be persisted over time. In the Cloud, because the underlying infrastructure can be moved or removed, it is not possible to make such assumptions. Instead of using the file system, store your temporary information in a dedicated Cloud-based storage or in a NoSQL database.

Criticality

BLOCKER

MEDIUM

Migration Impacts

CODE | FRAMEWORK | ARCHITECTURE

Files list

```
\path\to\file1
\path\to\file2
\path\to\file3
```

Searched Code Patterns

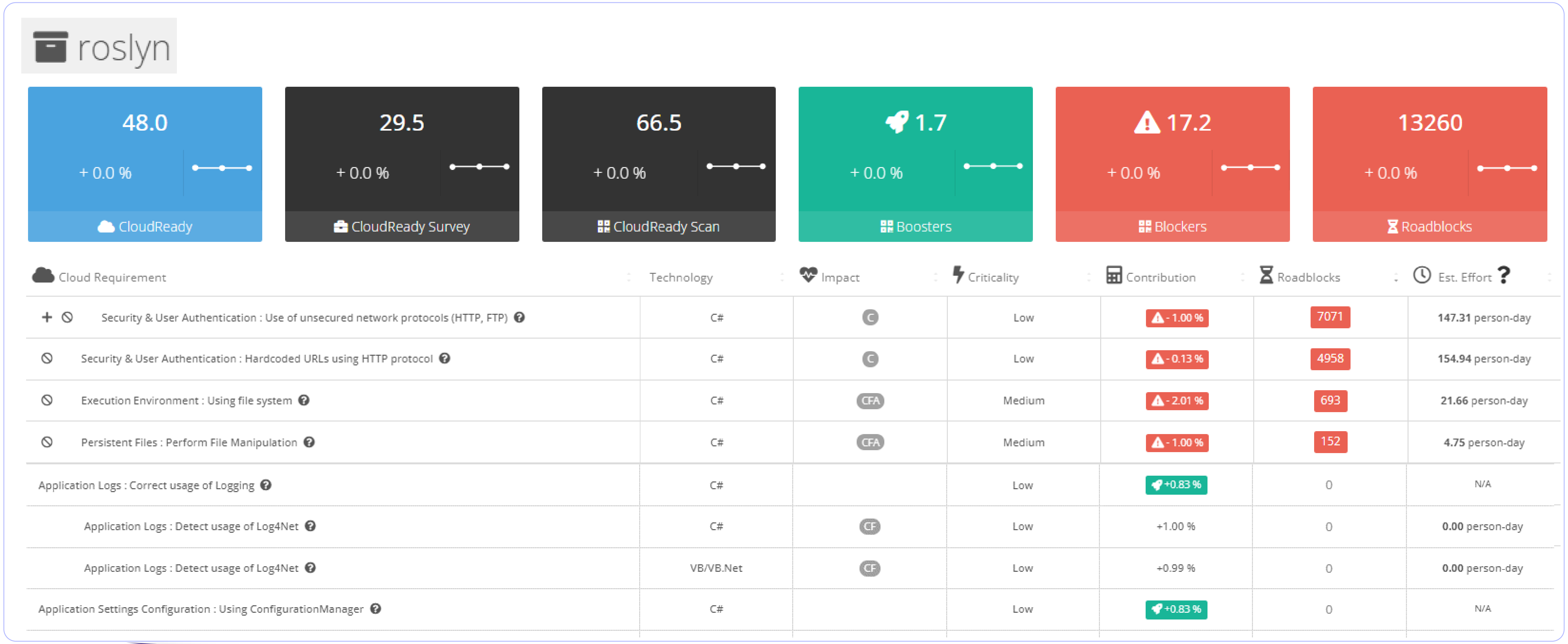
For Java applications:

`import org.apache.commons.io.FileUtils;` or `import java.io.File;`

and `moveFile()` or `forceDelete()` or `deleteQuitely()` or `copyFile()` or `write()`

Each Blocker is described in detail including remediation advice.

Cloud Boosters & Blockers for Roslyn Application



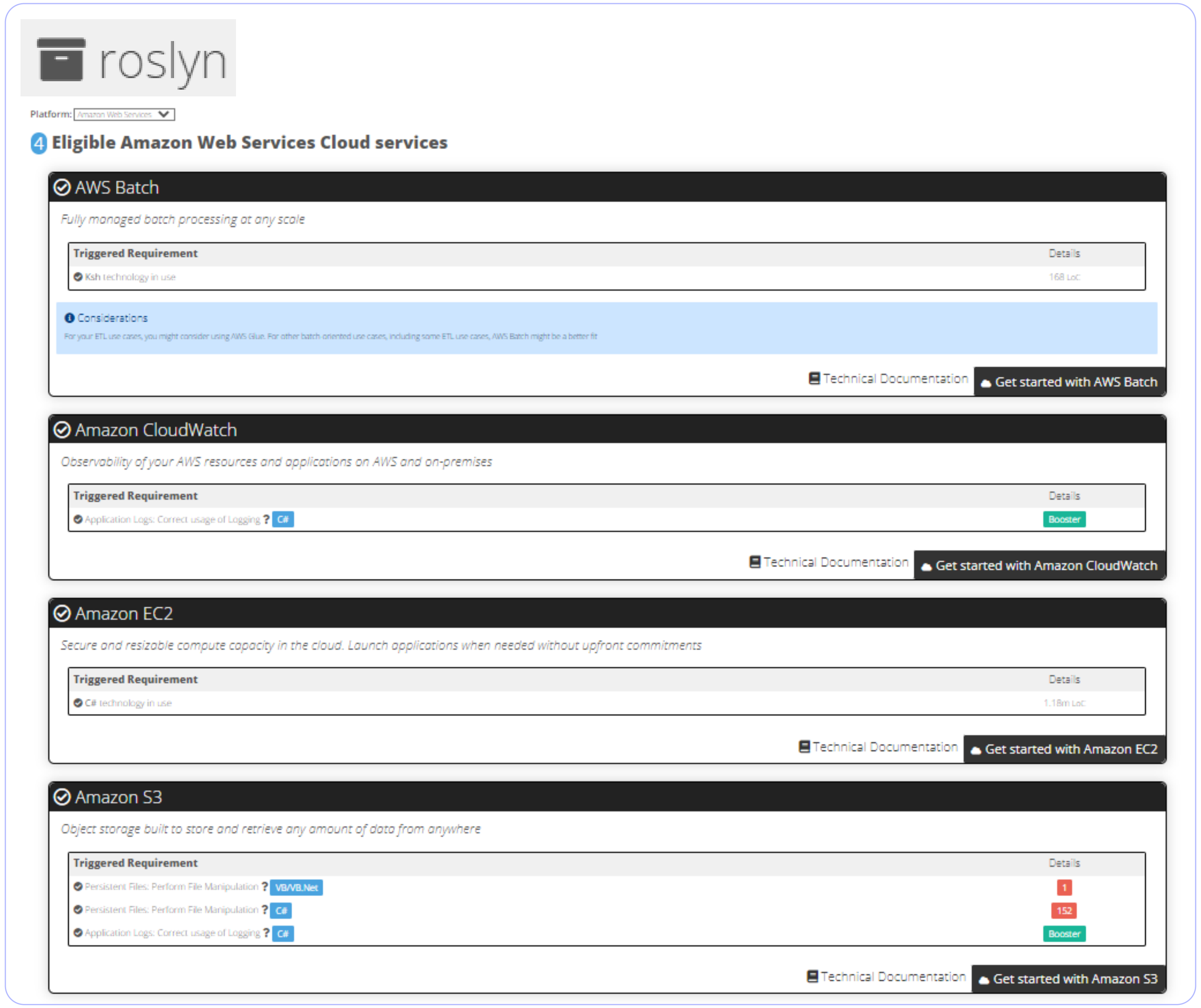
roslyn

CloudReady	CloudReady Survey	CloudReady Scan	Boosters	Blockers	Roadblocks
48.0	29.5	66.5	1.7	17.2	13260
+ 0.0 %	+ 0.0 %	+ 0.0 %	+ 0.0 %	+ 0.0 %	+ 0.0 %

Cloud Requirement	Technology	Impact	Criticality	Contribution	Roadblocks	Est. Effort ?
+ ⃠ Security & User Authentication : Use of unsecured network protocols (HTTP, FTP) ?	C#	C	Low	- 1.00 %	7071	**147.31** person-day
⃠ Security & User Authentication : Hardcoded URLs using HTTP protocol ?	C#	C	Low	- 0.13 %	4958	**154.94** person-day
⃠ Execution Environment : Using file system ?	C#	CFA	Medium	- 2.01 %	693	**21.66** person-day
⃠ Persistent Files : Perform File Manipulation ?	C#	CFA	Medium	- 1.00 %	152	**4.75** person-day
Application Logs : Correct usage of Logging ?	C#		Low	+0.83 %	0	N/A
Application Logs : Detect usage of Log4Net ?	C#	CF	Low	+1.00 %	0	**0.00** person-day
Application Logs : Detect usage of Log4Net ?	VB/VB.Net	CF	Low	+0.99 %	0	**0.00** person-day
Application Settings Configuration : Using ConfigurationManager ?	C#		Low	+0.83 %	0	N/A

Insights are available at the application level to understand the specific Blockers that occur within each application and estimated effort to remove them so that the modernization plan can be further refined based on individual application characteristics.

Cloud Native Service Recommendations for Roslyn Application



roslyn

Platform: Amazon Web Services

4 Eligible Amazon Web Services Cloud services

AWS Batch

Fully managed batch processing at any scale

Triggered Requirement	Details
Ksh technology in use	168 LoC

Considerations

For your ETL use cases, you might consider using AWS Glue. For other batch oriented use cases, including some ETL use cases, AWS Batch might be a better fit

Amazon CloudWatch

Observability of your AWS resources and applications on AWS and on-premises

Triggered Requirement	Details
Application Logs: Correct usage of Logging ? C#	Booster

Amazon EC2

Secure and resizable compute capacity in the cloud. Launch applications when needed without upfront commitments

Triggered Requirement	Details
C# technology in use	1.18m LoC

Amazon S3

Object storage built to store and retrieve any amount of data from anywhere

Triggered Requirement	Details
Persistent Files: Perform File Manipulation ? VB/VB.Net	1
Persistent Files: Perform File Manipulation ? C#	152
Application Logs: Correct usage of Logging ? C#	Booster

Specific cloud native services on AWS, Azure, Google Cloud, Oracle Cloud, or IBM Cloud are recommended based on each application's technical characteristics.

Cloud Maturity Recommendations

Applications to **Refactor** for PaaS (less effort):

- Roslyn, Shopizer, Unicorn, Product Management, IMDB, Budget, MultiCloud, Mando, Starlord

Applications to **Rearchitect** for PaaS (medium effort):

- Hadoop

Applications to **Rebuild** for PaaS (most effort):

- GCP-Client

Applications to **Retire**:

- Cassandra, Groot

Top cloud native services to adopt on AWS:

- AWS Batch, Amazon EC2, Amazon ECS, Amazon EKS, Amazon S3

The cloud native adoption recommendations are then summarized to develop the overall roadmap for the portfolio.

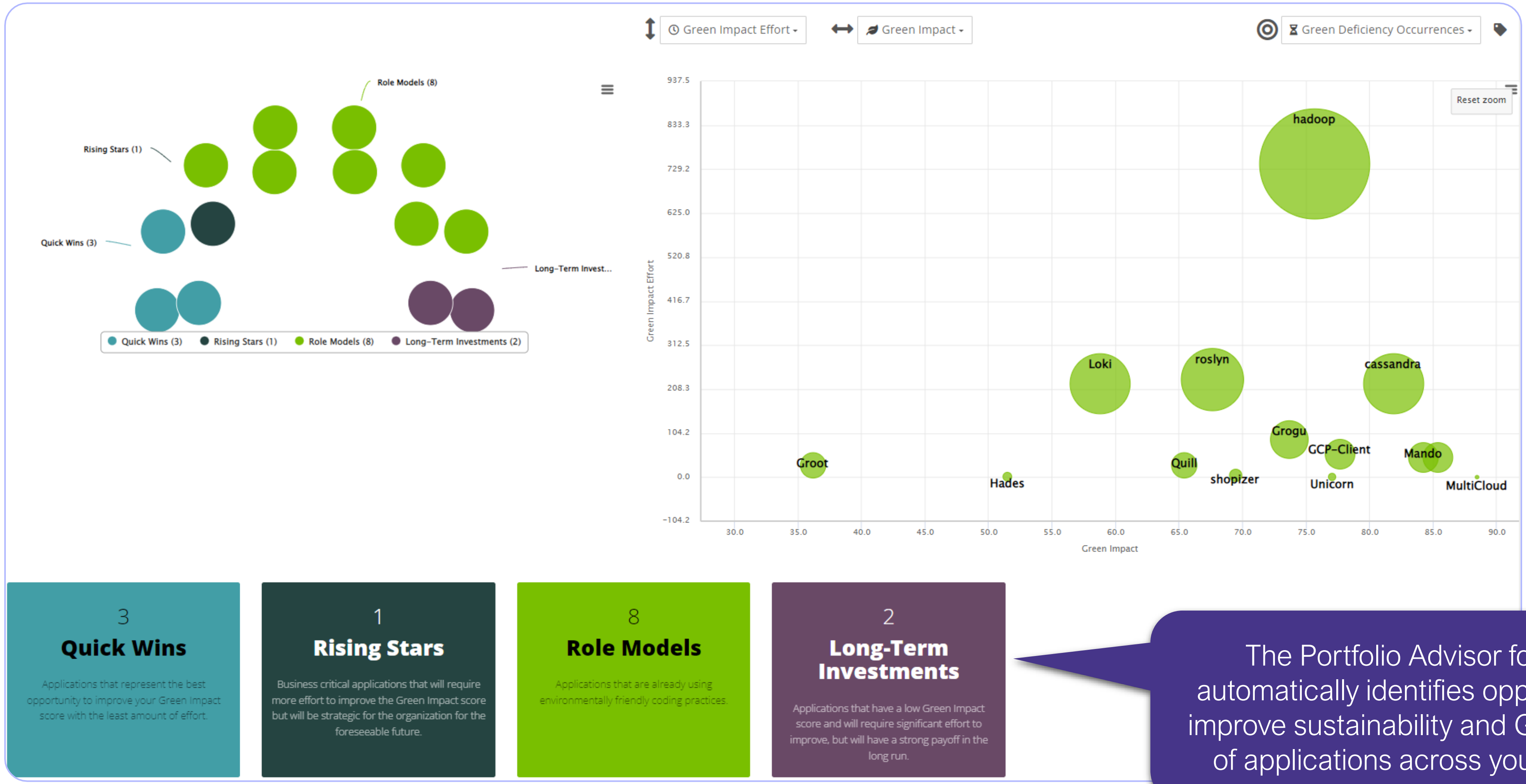
Green Impact

Green Impact Section

This section of the report contains key insights generated by CAST Highlight on the Green Impact of applications that should be improved and tracked over time including:

- Prioritized actions to take for applications to improve green impact
- Green Deficiencies in the code, estimated effort to remove them, and the required code changes
- A view of the Green Impact score trends over time
- Summarized action plan for the application portfolio

Portfolio Advisor for Green



3
Quick Wins
Applications that represent the best opportunity to improve your Green Impact score with the least amount of effort.

1
Rising Stars
Business critical applications that will require more effort to improve the Green Impact score but will be strategic for the organization for the foreseeable future.

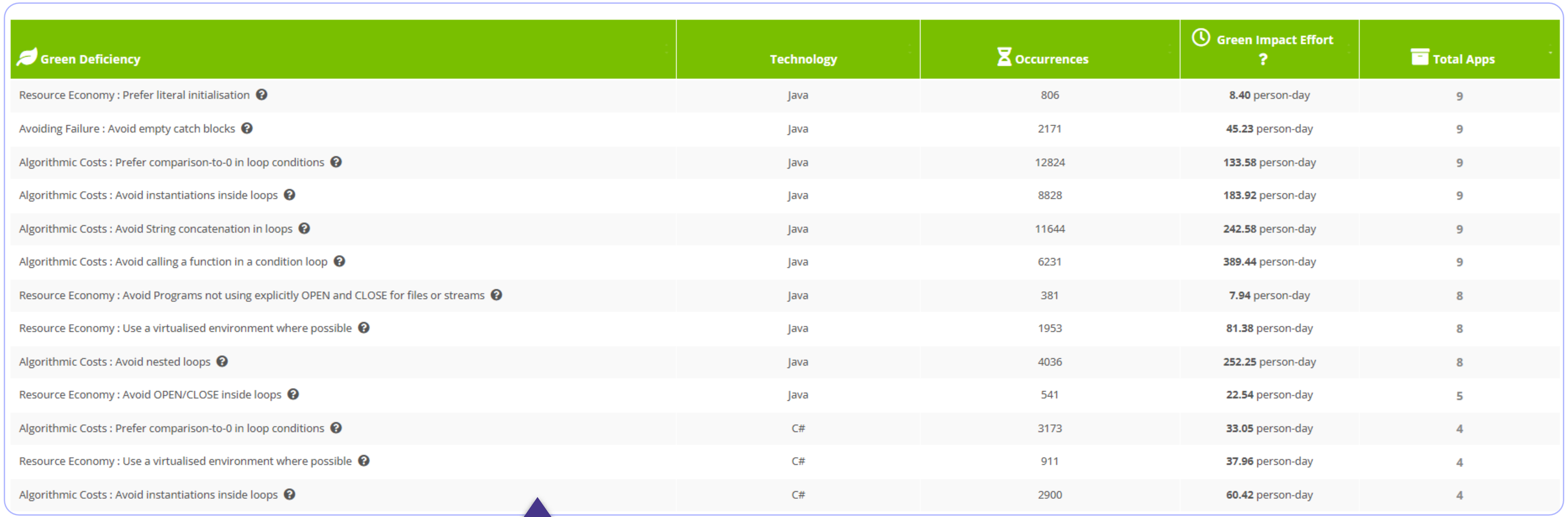
8
Role Models
Applications that are already using environmentally friendly coding practices.

2
Long-Term Investments
Applications that have a low Green Impact score and will require significant effort to improve, but will have a strong payoff in the long run.

The Portfolio Advisor for Green automatically identifies opportunities to improve sustainability and Green Impact of applications across your portfolio.

CAST

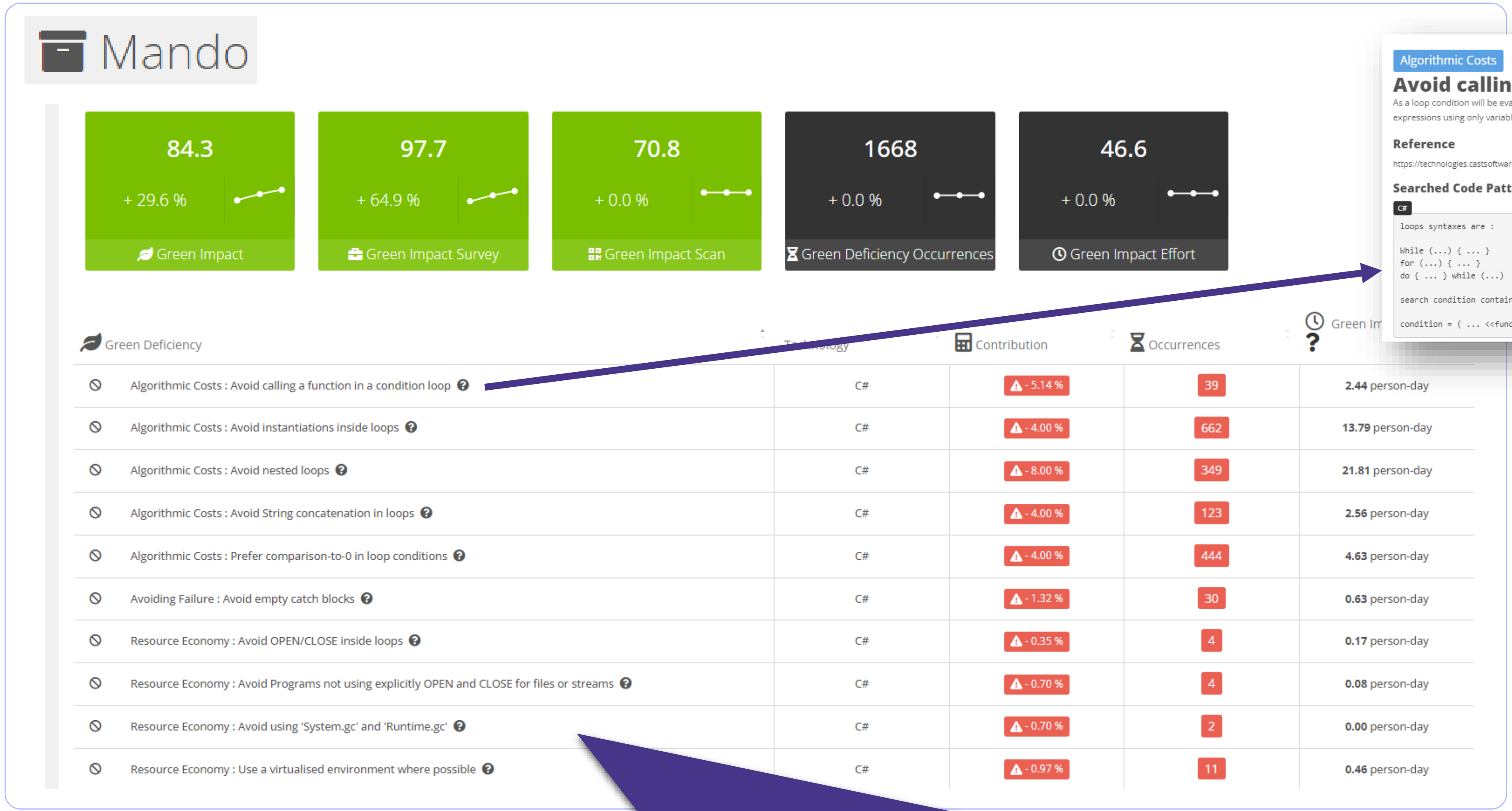
Green Deficiencies



Green Deficiency	Technology	Occurrences	Green Impact Effort ?	Total Apps
Resource Economy : Prefer literal initialisation	Java	806	**8.40** person-day	9
Avoiding Failure : Avoid empty catch blocks	Java	2171	**45.23** person-day	9
Algorithmic Costs : Prefer comparison-to-0 in loop conditions	Java	12824	**133.58** person-day	9
Algorithmic Costs : Avoid instantiations inside loops	Java	8828	**183.92** person-day	9
Algorithmic Costs : Avoid String concatenation in loops	Java	11644	**242.58** person-day	9
Algorithmic Costs : Avoid calling a function in a condition loop	Java	6231	**389.44** person-day	9
Resource Economy : Avoid Programs not using explicitly OPEN and CLOSE for files or streams	Java	381	**7.94** person-day	8
Resource Economy : Use a virtualised environment where possible	Java	1953	**81.38** person-day	8
Algorithmic Costs : Avoid nested loops	Java	4036	**252.25** person-day	8
Resource Economy : Avoid OPEN/CLOSE inside loops	Java	541	**22.54** person-day	5
Algorithmic Costs : Prefer comparison-to-0 in loop conditions	C#	3173	**33.05** person-day	4
Resource Economy : Use a virtualised environment where possible	C#	911	**37.96** person-day	4
Algorithmic Costs : Avoid instantiations inside loops	C#	2900	**60.42** person-day	4

The Green Deficiency patterns in the code that contribute to excess resource utilization and energy consumption are identified across the portfolio including number of occurrences, effort to remediate, and the specific applications where they occur.

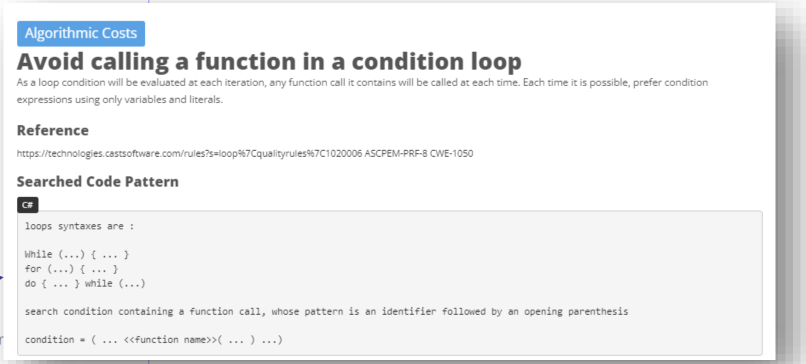
Green Deficiencies Detail for Mando Application



Mando

84.3	97.7	70.8	1668	46.6
+ 29.6 %	+ 64.9 %	+ 0.0 %	+ 0.0 %	+ 0.0 %
Green Impact	Green Impact Survey	Green Impact Scan	Green Deficiency Occurrences	Green Impact Effort

Green Deficiency	Technology	Contribution	Occurrences	Green Im
Algorithmic Costs : Avoid calling a function in a condition loop	C#	- 5.14 %	39	2.44 person-day
Algorithmic Costs : Avoid instantiations inside loops	C#	- 4.00 %	662	13.79 person-day
Algorithmic Costs : Avoid nested loops	C#	- 8.00 %	349	21.81 person-day
Algorithmic Costs : Avoid String concatenation in loops	C#	- 4.00 %	123	2.56 person-day
Algorithmic Costs : Prefer comparison-to-0 in loop conditions	C#	- 4.00 %	444	4.63 person-day
Avoiding Failure : Avoid empty catch blocks	C#	- 1.32 %	30	0.63 person-day
Resource Economy : Avoid OPEN/CLOSE inside loops	C#	- 0.35 %	4	0.17 person-day
Resource Economy : Avoid Programs not using explicitly OPEN and CLOSE for files or streams	C#	- 0.70 %	4	0.08 person-day
Resource Economy : Avoid using 'System.gc' and 'Runtime.gc'	C#	- 0.70 %	2	0.00 person-day
Resource Economy : Use a virtualised environment where possible	C#	- 0.97 %	11	0.46 person-day



Algorithmic Costs

Avoid calling a function in a condition loop

As a loop condition will be evaluated at each iteration, any function call it contains will be called at each time. Each time it is possible, prefer condition expressions using only variables and literals.

Reference

https://technologies.castsoftware.com/rules?s=loop%7Cqualityrules%7C1020006 ASCPEM-PRF-8 CWE-1050

Searched Code Pattern

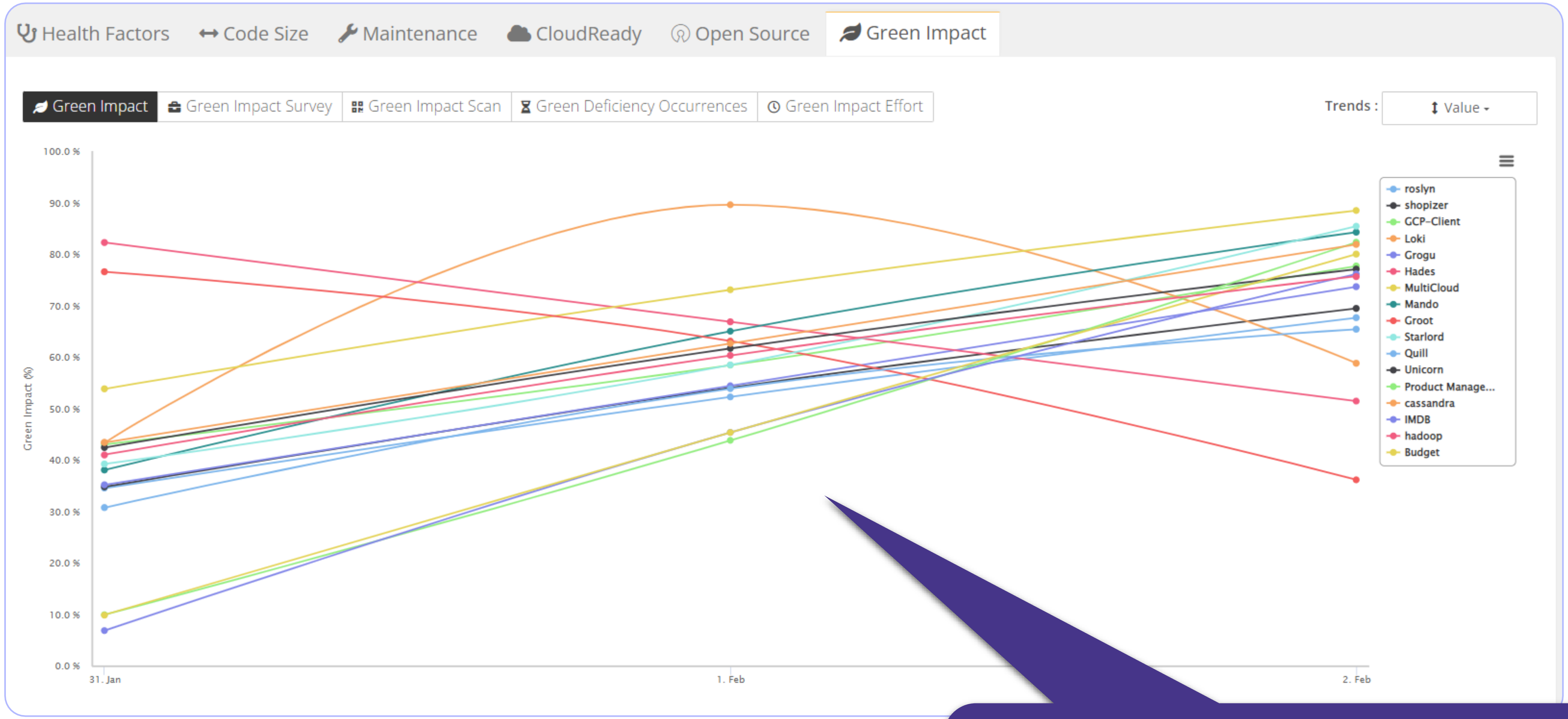
C#

```
loops syntaxes are :

While (...) { ... }
for (...) { ... }
do { ... } while (...)

search condition containing a function call, whose pattern is an identifier followed by an opening parenthesis

condition = ( ... <<function name>>( ... ) ...)
```

Insights are available at the application level to understand the specific Green Deficiencies that occur within each application, estimated effort to remove them, and remediation advice so that applications can be made more sustainable.

CAST

Green Impact Trends



Applications are continuously monitored to view progress being made on green impact (and other metrics) across all applications.

Green Impact Recommendations

Shopizer: Remove the top 10 Green Deficiencies, less than one week of estimated effort

Quill: Remove top 2 Green Deficiencies, less than two weeks of estimated effort

Mando: Remove top Green Deficiency, two weeks of estimated effort

Applications to address in the future:

- Groot
- Roslyn
- Grogu

Review two “Role Model” applications to identify best practices to share across the team:

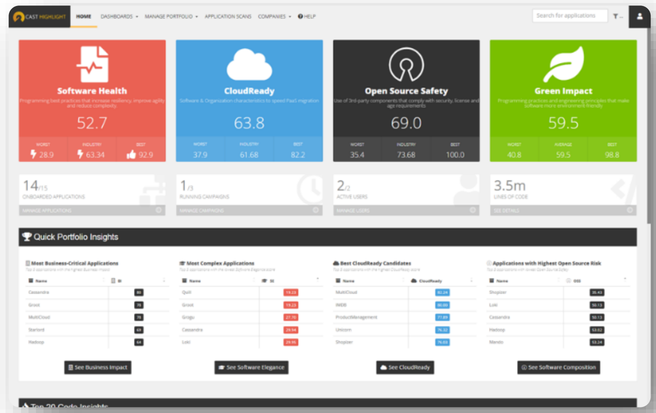
- MultiCloud
- Starlord

Specific recommendations on how to improve Green Impact are summarized.

Why CAST Highlight?

CAST Highlight gives enterprise leaders rapid insights across entire portfolios. Automated source code analysis with built-in surveys for business context. Portfolio views. Instant drilldowns. Recommendations. Operational in a week. Across hundreds of applications.

- **Automate** Portfolio Governance
- **Manage** Open Source Risk
- **Accelerate** Cloud Migration
- **Improve** Green Impact



Software Health
Resiliency
Agility
Technical Debt

Cloud Maturity
Roadmaps
Blockers & Effort
Cloud Native Services

Software Composition
OSS Vulnerabilities
OSS IP / Licensing Risks
SBOM

Green Impact
Deficiencies
Remediation Advice
Trends

Trusted By:

BCG · WELLS FARGO · Microsoft · accenture · BNY MELLON · IBM · UNITED STATES AIR FORCE · BMW

CAST

CAST

See everything, advance anything

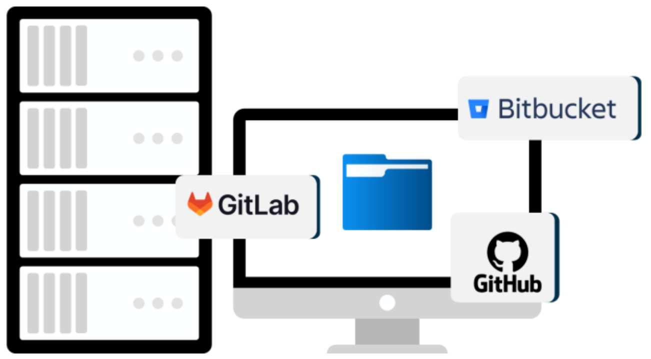
Request demo

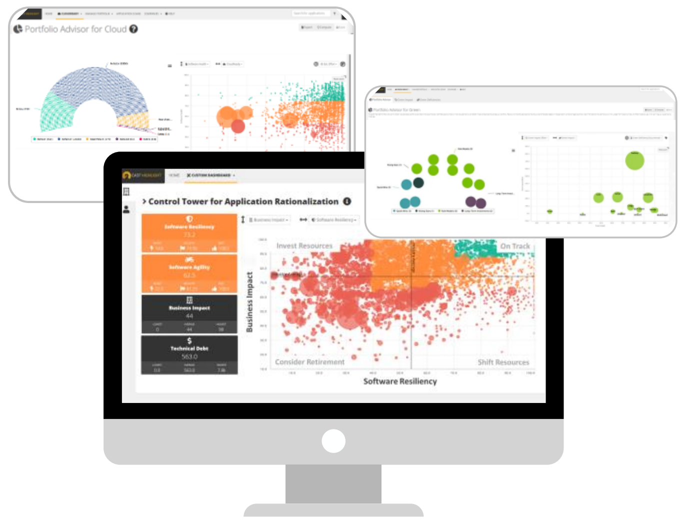
castsoftware.com

Appendix

Data collection for CAST Highlight

Simple 3-step process – rapid to implement, easy to use, easy to scale and integrate





Step 1

Code reader analyzes source code automatically from the repositories with predefined frequency.

Step 2

Encrypted intelligence is uploaded to secure cloud (27001-certified). No code leaves premises.

Step 3

Instant visibility via dashboards, heat maps, charts, recommendations. API-based data integration.

Key Metrics & Methodology Definitions

Key Metric	Description	Direct Interpretation	Business Impact
Cloud Maturity	Measure of software and organization characteristics to speed PaaS migration	Significant number of roadblocks found that could slow down a Cloud migration	Opportunity to reduce cost, increase elasticity and embrace innovation
Software Resiliency	Measure the robustness and how bullet-proof is the Software against production failure	Reflects presence of code patterns that may comprise vulnerability of the software	Customer Satisfaction Customer Confidence / Loyalty Opportunities & Revenue
Software Agility	Measure to indicate the easiness of a development team to understand and maintain an application	Reflects absence of embedded documentation and code readability good practices	Maintenance Cost Transferability
Software Elegance	Measures the ability to deliver software value with less code complexity	Indicates decreased quality in code, resulting in higher defects that become costly to fix	Time to Market Innovation
Open Source Safety	Measure risk associated with the use of 3rd-party components that comply security, license, and age requirements.	Analysis of open-source and 3rd-party components in use that could include security vulnerabilities, risky licensing requirements, or obsolete technology.	Reduce security risk, reduce legal exposure, reduce operational risk
Green Impact	Measure programming practices and engineering principles that make software more environmentally-friendly.	Identification of Green Deficiency patterns in the code of applications that contribute to excess resource utilization and energy consumption.	Support ESG requirements, make software greener, more resilient, less expensive, and more performant