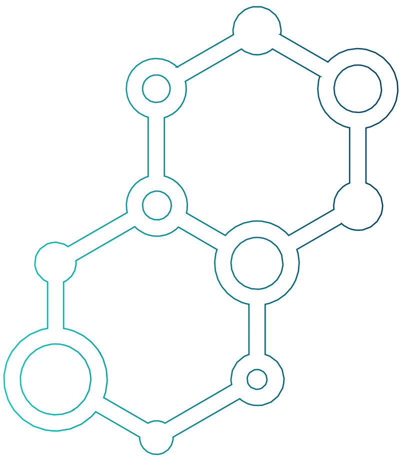
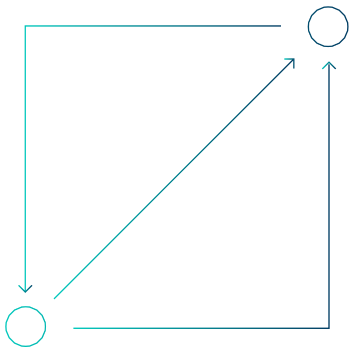
Sealights

BEST PRACTICES FOR IMPLEMENTING SOFTWARE QUALITY INTELLIGENCE WITH SEALIGHTS

TABLE OF CONTENTS



INTRODUCTION

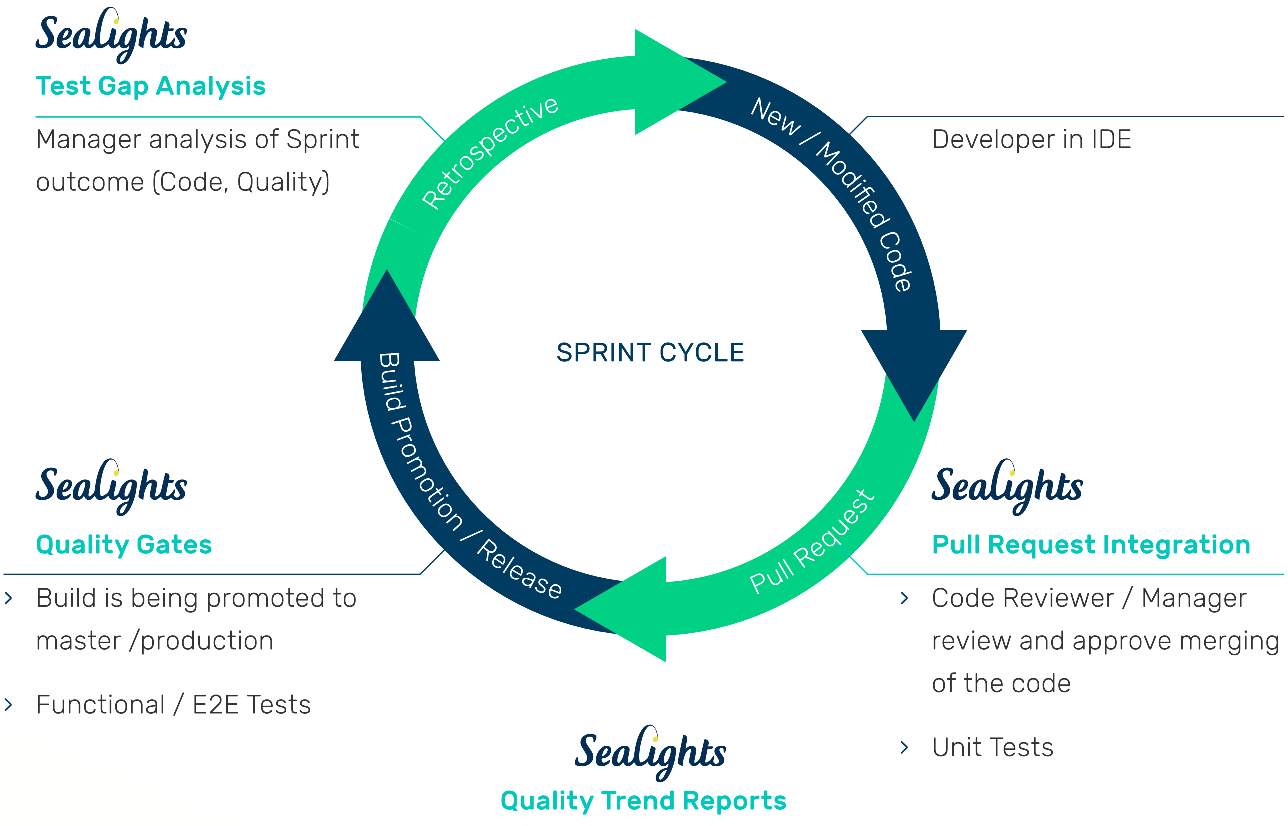


Whilst changes should always be regarded as positive, often change can become the enemy of quality if not executed and managed rigorously. That is why at SeaLights we focus on providing you with the ability to capture and prioritise changes in your SDLC, but at the same time avoid the unnecessary time, effort, and cost of managing things that are not impacted by changes.

There is a widely used business adage “If you can’t measure it, you can’t improve it” which should only be applied sparingly to avoid another business adage “analysis paralysis”. Fortunately, SeaLights provides constant measurement metrics in real-time that can be consumed as and when required.

In this section we will review the SeaLights best practices for how to measure the impact of changes to your applications and improve the quality of what is delivered. This assumes that you already have completed the steps to integrate SeaLights with your CI (Build Scanner) and testing frameworks (Test Listeners).

TYPICAL SPRINT CYCLE WITH SEALIGHTS



We will break up the 6 steps that we recommend you follow to minimize quality risks into the following logical phases, for specific personas, aligned with a typical sprint cycle as shown above:

- Identify New/Modified Code in the background without impacting existing development
- Guide the software development team to the best decisions
- Determine release readiness, identify and prevent untested code changes making it to production
- Direct teams to where to develop and execute the minimal number of tests

STEP 1 – IDENTIFY NEW/MODIFIED CODE

1

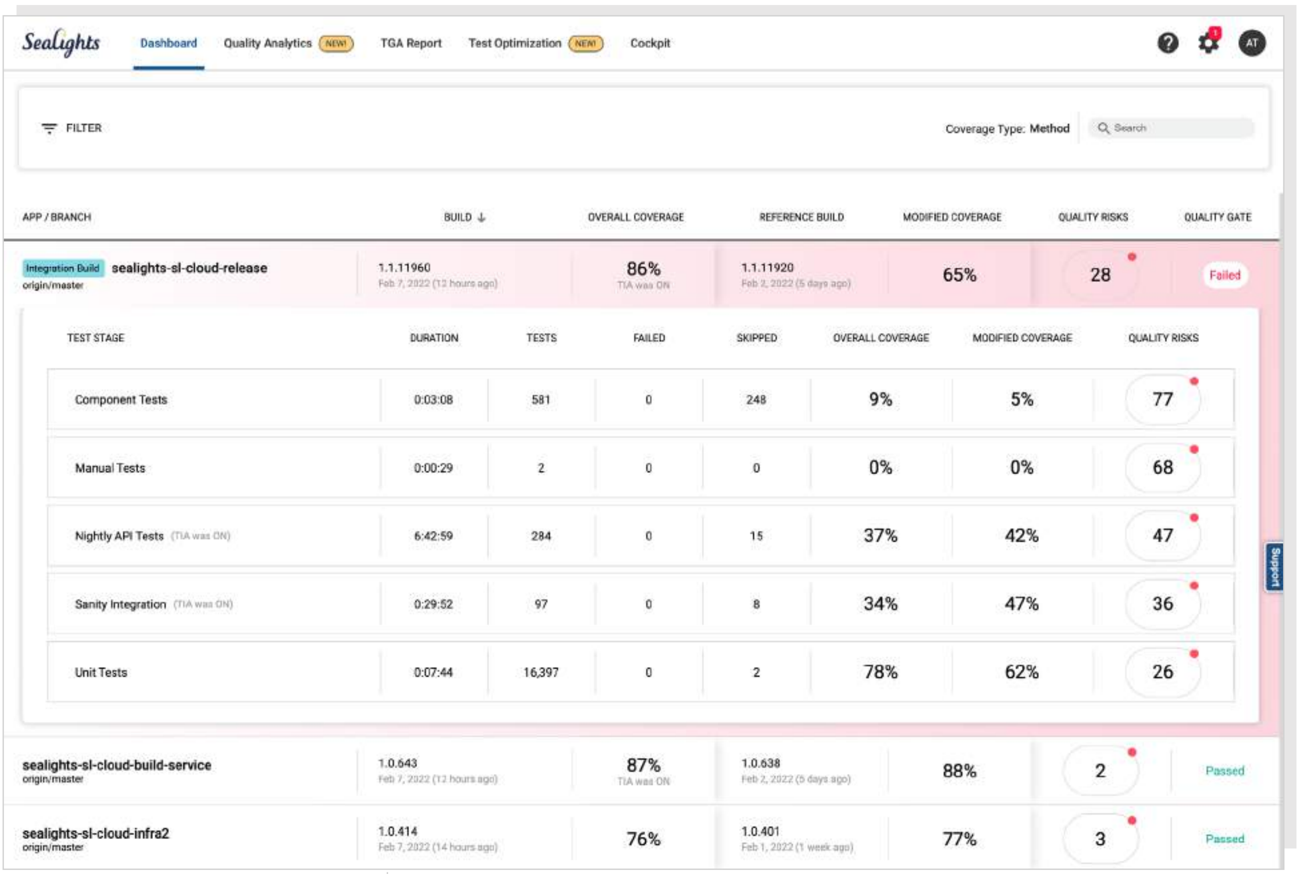
The first step, or "Line of Defence", in implementing a Software Quality Intelligence process involves putting in place a consistent methodology that will enable you to capture code coverage across all test stages (unit, component, functional, integration, sanity, API, user acceptance, manual, etc.) and start the process of identifying quality risks.

PERSONAS AND CADENCE

Typically the best practices that will be developed and implemented in this step would be led by, but not be limited to, the Quality Engineering organization. Once established we recommend that the scope of the scanned code is validated for every new application and service, new code labels are created to identify new/modified teams and functional areas. Reference Builds should also be regularly updated as the lifecycle of the application progresses.

FUNCTIONAL OVERVIEW

Before we review the process let's look at the SeaLights functionality that enables this, starting with the SeaLights Dashboard.



Coverage Type: Method

APP / BRANCH	BUILD ↓	OVERALL COVERAGE	REFERENCE BUILD	MODIFIED COVERAGE	QUALITY RISKS	QUALITY GATE
Integration Build sealights-sl-cloud-release origin/master	1.1.11960 Feb 7, 2022 (12 hours ago)	86% TIA was ON	1.1.11920 Feb 2, 2022 (5 days ago)	65%	28	Failed
sealights-sl-cloud-build-service origin/master	1.0.643 Feb 7, 2022 (12 hours ago)	87% TIA was ON	1.0.638 Feb 2, 2022 (5 days ago)	88%	2	Passed
sealights-sl-cloud-infra2 origin/master	1.0.414 Feb 7, 2022 (14 hours ago)	76%	1.0.401 Feb 1, 2022 (1 week ago)	77%	3	Passed

TEST STAGE	DURATION	TESTS	FAILED	SKIPPED	OVERALL COVERAGE	MODIFIED COVERAGE	QUALITY RISKS
Component Tests	0:03:08	581	0	248	9%	5%	77
Manual Tests	0:00:29	2	0	0	0%	0%	68
Nightly API Tests (TIA was ON)	6:42:59	284	0	15	37%	42%	47
Sanity Integration (TIA was ON)	0:29:52	97	0	8	34%	47%	36
Unit Tests	0:07:44	16,397	0	2	78%	62%	26

Figure 1: SeaLights Dashboard

Collation of data from multiple sources across the software development pipeline, such as code change, test stages, code execution, CI tools, production data, historical build information, and more populates the SeaLights Dashboard under the following categories:

- Overall Coverage
- Modified Coverage

Overall Coverage identifies the percentage of methods that have been tested by one or more test stages using the simple formula = *number of tested methods/overall number of methods.*

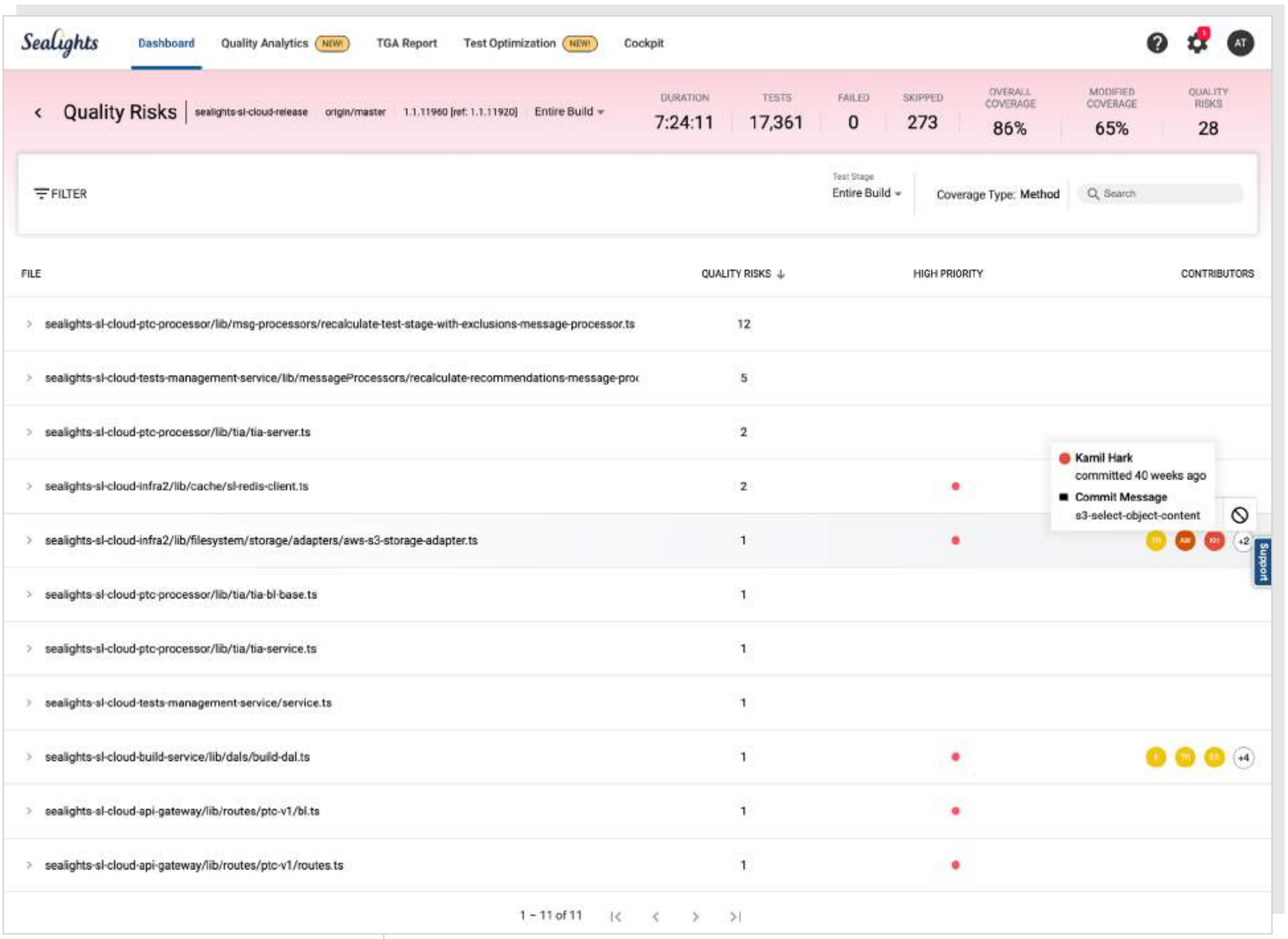
> **Example:** If you had an application or service that contained a total of 100 methods and 70 of those methods had been tested by one or more test stages then the overall coverage would be 70%.

Modified Coverage identifies the percentage of **new or modified** methods that have been tested by one or more test stages using the simple formula = *number of tested new or modified methods/ number of new or modified methods.*

> **Example:** If you modified 10 of the 100 methods, and if 8 of those modified methods were tested by one or more test stages, then the modified coverage would be 80%.

New or modified methods that have not been tested by any test stage represent a **Quality Risk**.

Example: The 2 new or modified methods that were not tested by any test stage in the example above would appear as 2 quality risks in the SeaLights Dashboard.



Quality Risks | sealights-sl-cloud-release origin/master 1.1.11960 [ref: 1.1.11920] Entire Build

DURATION	TESTS	FAILED	SKIPPED	OVERALL COVERAGE	MODIFIED COVERAGE	QUALITY RISKS
7:24:11	17,361	0	273	86%	65%	28

FILTER — Test Stage: Entire Build — Coverage Type: Method — Search

FILE	QUALITY RISKS ↓	HIGH PRIORITY	CONTRIBUTORS
sealights-sl-cloud-ptc-processor/lib/msg-processors/recalculate-test-stage-with-exclusions-message-processor.ts	12		
sealights-sl-cloud-tests-management-service/lib/messageProcessors/recalculate-recommendations-message-pro	5		
sealights-sl-cloud-ptc-processor/lib/tia/tia-server.ts	2		
sealights-sl-cloud-infra2/lib/cache/sl-redis-client.ts	2	●	
sealights-sl-cloud-infra2/lib/filesystem/storage/adapters/aws-s3-storage-adapter.ts	1	●	TH KW KH +2
sealights-sl-cloud-ptc-processor/lib/tia/tia-bl-base.ts	1		
sealights-sl-cloud-ptc-processor/lib/tia/tia-service.ts	1		
sealights-sl-cloud-tests-management-service/service.ts	1		
sealights-sl-cloud-build-service/lib/dals/build-dal.ts	1	●	+4
sealights-sl-cloud-api-gateway/lib/routes/ptc-v1/bl.ts	1	●	
sealights-sl-cloud-api-gateway/lib/routes/ptc-v1/routes.ts	1	●	

Kamil Hark
committed 40 weeks ago
Commit Message
s3-select-object-content

1 – 11 of 11

Figure 2: Quality Risks

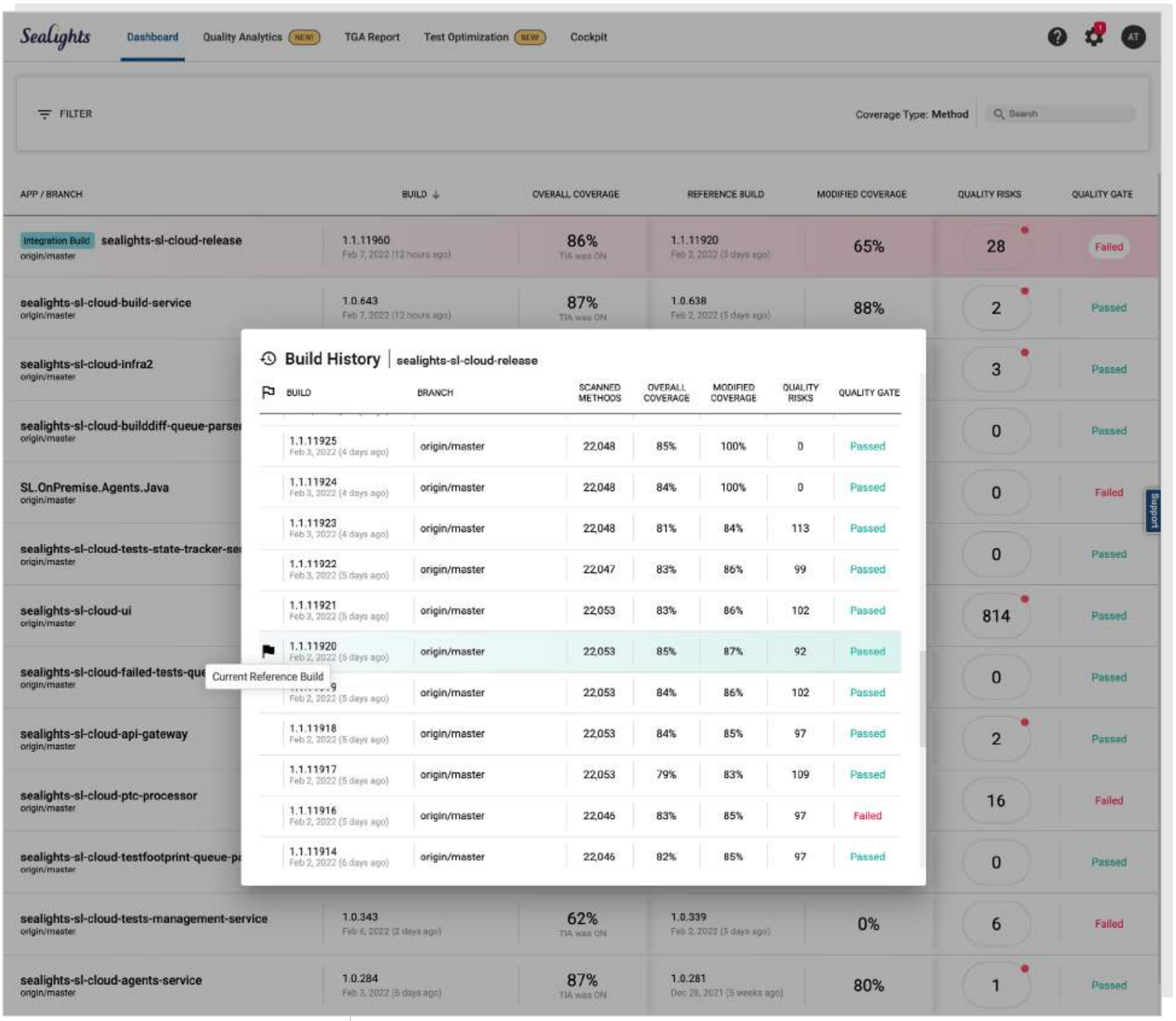
New or modified methods are determined by comparing the latest build with a Reference Build. By default, the Reference Build is the build prior to the current build in each branch. It can, however, be set (or changed) by the user by hovering over the build in the build history that you wish to specify as the Reference Build and setting it with the flag icon.

Best practice for setting the Reference Build is dependent upon where you are in the lifecycle of the application but here are some recommendations:

- The first build from the current sprint
- The last build from the previous sprint
- The last build promoted to the next branch (e.g. feature-branch to develop to main)
- The last build from the previous sprint
- The last build pushed to Production

Important: The Reference Build setting will only apply after the next build.

This is displayed in *Figure 3: Defining the Reference Build* below.



APP / BRANCH	BUILD ↓	OVERALL COVERAGE	REFERENCE BUILD	MODIFIED COVERAGE	QUALITY RISKS	QUALITY GATE
Integration Build sealights-sl-cloud-release origin/master	1.1.11960 Feb 7, 2022 (12 hours ago)	86% TIA was ON	1.1.11920 Feb 2, 2022 (5 days ago)	65%	28	Failed
sealights-sl-cloud-build-service origin/master	1.0.643 Feb 7, 2022 (12 hours ago)	87% TIA was ON	1.0.638 Feb 2, 2022 (5 days ago)	88%	2	Passed
sealights-sl-cloud-infra2 origin/master					3	Passed
sealights-sl-cloud-builddiff-queue-parser origin/master					0	Passed
SL.OnPremise.Agents.Java origin/master					0	Failed
sealights-sl-cloud-tests-state-tracker-se origin/master					0	Passed
sealights-sl-cloud-ui origin/master					814	Passed
sealights-sl-cloud-failed-tests-que origin/master					0	Passed
sealights-sl-cloud-api-gateway origin/master					2	Passed
sealights-sl-cloud-ptc-processor origin/master					16	Failed
sealights-sl-cloud-testfootprint-queue-p origin/master					0	Passed
sealights-sl-cloud-tests-management-service origin/master	1.0.343 Feb 6, 2022 (2 days ago)	62% TIA was ON	1.0.339 Feb 2, 2022 (5 days ago)	0%	6	Failed
sealights-sl-cloud-agents-service origin/master	1.0.284 Feb 3, 2022 (5 days ago)	87% TIA was ON	1.0.281 Dec 28, 2021 (5 weeks ago)	80%	1	Passed

Build History | sealights-sl-cloud-release

⚐	BUILD	BRANCH	SCANNED METHODS	OVERALL COVERAGE	MODIFIED COVERAGE	QUALITY RISKS	QUALITY GATE
	1.1.11925 Feb 3, 2022 (4 days ago)	origin/master	22,048	85%	100%	0	Passed
	1.1.11924 Feb 3, 2022 (4 days ago)	origin/master	22,048	84%	100%	0	Passed
	1.1.11923 Feb 3, 2022 (4 days ago)	origin/master	22,048	81%	84%	113	Passed
	1.1.11922 Feb 3, 2022 (5 days ago)	origin/master	22,047	83%	86%	99	Passed
	1.1.11921 Feb 3, 2022 (5 days ago)	origin/master	22,053	83%	86%	102	Passed
⚑	1.1.11920 Feb 2, 2022 (5 days ago)	origin/master	22,053	85%	87%	92	Passed
	[illegible]9 Feb 2, 2022 (5 days ago)	origin/master	22,053	84%	86%	102	Passed
	1.1.11918 Feb 2, 2022 (5 days ago)	origin/master	22,053	84%	85%	97	Passed
	1.1.11917 Feb 2, 2022 (5 days ago)	origin/master	22,053	79%	83%	109	Passed
	1.1.11916 Feb 2, 2022 (5 days ago)	origin/master	22,046	83%	85%	97	Failed
	1.1.11914 Feb 2, 2022 (6 days ago)	origin/master	22,046	82%	85%	97	Passed

Current Reference Build

Figure 3: Defining the Reference Build

You also have the option to define **Ignored Code** to exclude code that is either auto-generated, getters, setters, default constructors, deprecated code, and other irrelevant code areas such as third-party code. Ignored Code will be removed from the Quality Risk and Coverage analysis processes.

Code Labels allow you to divide your code into areas of interest by defining categories representing either teams (Development, QA, etc), Functional Areas (Login, Reports, etc) or any division you'd like to see within your codebase. Furthermore, under each category, you can define labels that will represent the code area relevant to that label. When using the SeaLights dashboard you can filter your view according to these categories and labels defined here. You can also define labels as "High Priority" so Quality Risks around that label will be highlighted.

Important: The code labels may be applied to the entire application, Classes, Folder, File paths, and Files. However, we strongly recommend that you DO NOT apply code labels at the files level as it will not include new files that are added to the Folder/Class.will only apply after the next build.

For applications that consist of multiple components or services, you will report these components individually, before grouping them as one application in an **Integration Build**, on which cross-component tests will run against (see the example in *Figure 1: SeaLights Dashboard*).

IDENTIFY NEW / MODIFIED CODE PROCESS OVERVIEW

In this phase of the overall process, we have reviewed the best practice for using the SeaLights functionality to establish a consistent methodology to identify quality risks. To summarise:

1. Define the scope of the scanned code for the application, ignoring the irrelevant code
2. For the code in scope, define code labels for teams, functional areas, etc.
3. Set Reference Builds in line with where you are in your application development lifecycle
4. Where appropriate, create an Integration Build

Having implemented this consistent methodology, you can now use the data that has been gathered by SeaLights to effect increased collaboration across your teams and make better informed decisions based on the improved, centralized visibility of where your quality risks are on your journey to build a strategy to eradicate defects from your production applications.

STEP 2 – ANALYZE COVERAGE DATA

2

The second step, or "Line of Defence", in implementing a Software Quality Intelligence process involves applying the measurements gathered in the "Identify" phase to drive improved quality across all applications and services that are being reported to SeaLights through the following SeaLights features:

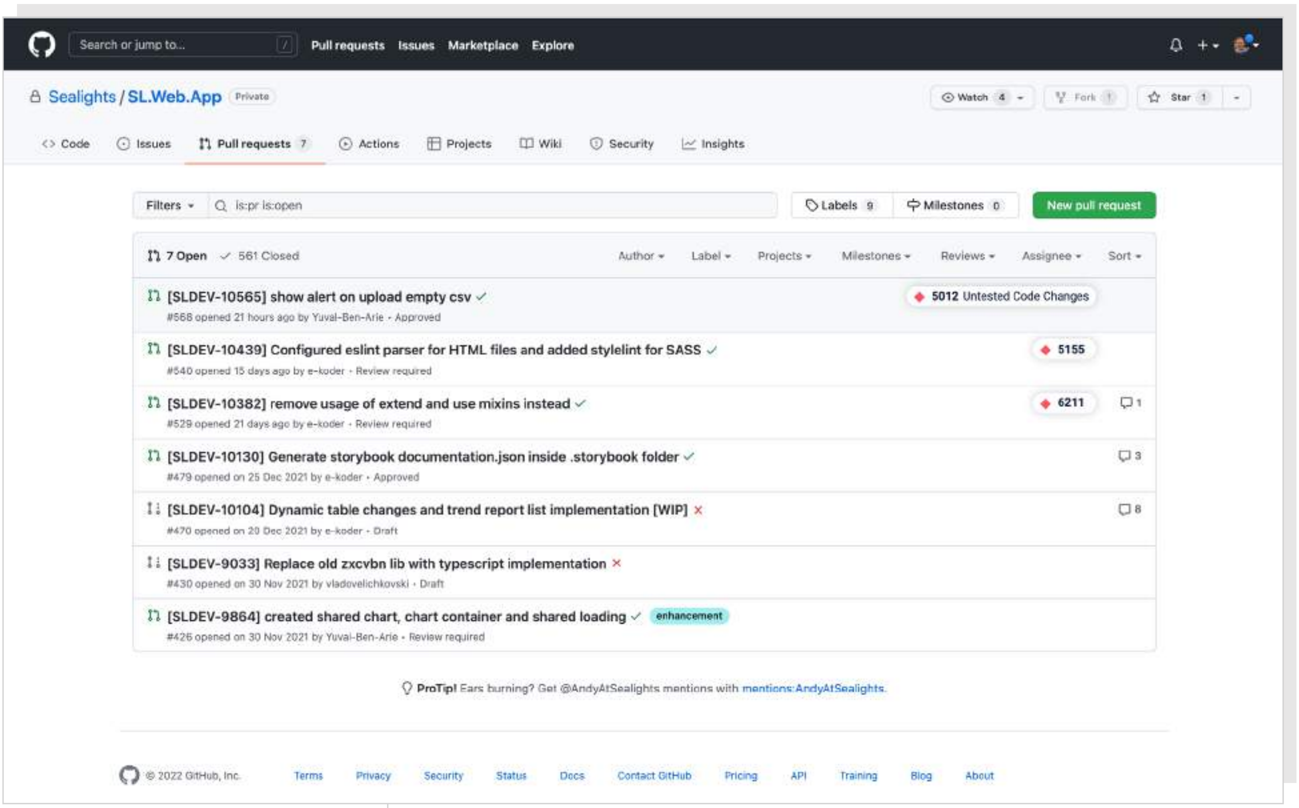
- Using the **Code Viewer** to view coverage data
- Analyzing **Quality Risks**

PERSONAS AND CADENCE

Code Reviewer functionality provided by the SeaLights Chrome Extension is typically used by the code reviewer or managers to review and comment on changes prior to approving the merge of the pull request. The review should be a collaborative process involving the software development and quality engineering teams who will look at the UCCs identified by SeaLights and then decide if the impact of the UCCs meet the repository's contributing guidelines and other quality standards.

FUNCTIONAL OVERVIEW

Before we review the process let's look at the SeaLights functionality that enables this.



7 Open ✓ 561 Closed

[SLDEV-10565] show alert on upload empty csv ✓ — 5012 Untested Code Changes
#568 opened 21 hours ago by Yuval-Ben-Arie • Approved

[SLDEV-10439] Configured eslint parser for HTML files and added stylelint for SASS ✓ — 5155
#540 opened 15 days ago by e-koder • Review required

[SLDEV-10382] remove usage of extend and use mixins instead ✓ — 6211 — 1
#529 opened 21 days ago by e-koder • Review required

[SLDEV-10130] Generate storybook documentation.json inside .storybook folder ✓ — 3
#479 opened on 25 Dec 2021 by e-koder • Approved

[SLDEV-10104] Dynamic table changes and trend report list implementation [WIP] × — 8
#470 opened on 20 Dec 2021 by e-koder • Draft

[SLDEV-9033] Replace old zxcvbn lib with typescript implementation ×
#430 opened on 30 Nov 2021 by vladovelichkovski • Draft

[SLDEV-9864] created shared chart, chart container and shared loading ✓ enhancement
#426 opened on 30 Nov 2021 by Yuval-Ben-Arie • Review required

ProTip! Ears burning? Get @AndyAtSealights mentions with mentions:AndyAtSealights.

Figure 4: Identifying Untested Code Changes with the SeaLights Chrome Extension (Pull Request list)

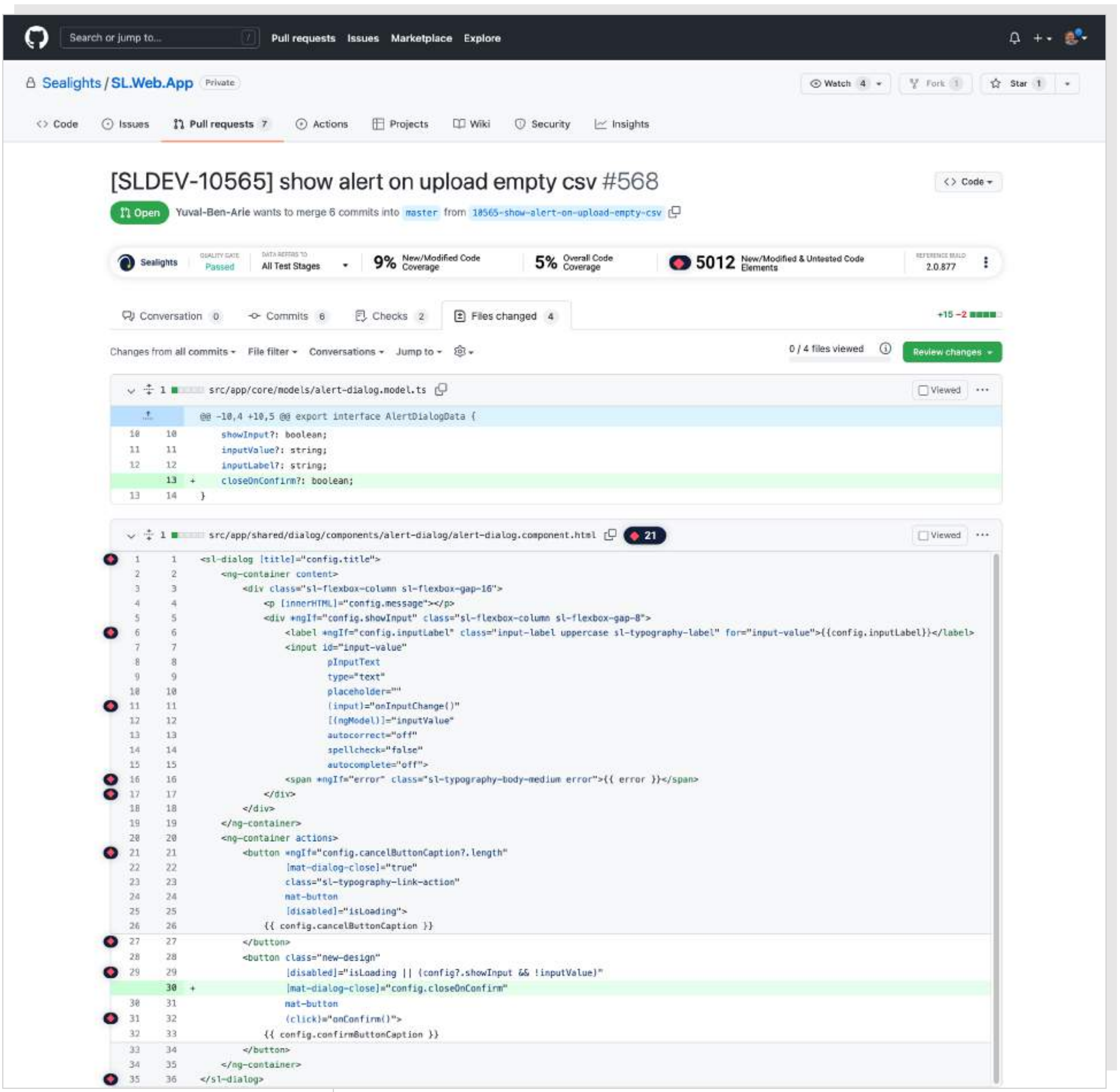
USING THE CODE VIEWER

The SeaLights Chrome Extension presents **Untested Code Changes** (UCCs) which may represent Quality Risks in an overlay over Pull Request lists and individual Pull Requests.

› The SeaLights Chrome Extension is supported on all popular Source Control Management (SCM) platforms, for example GitHub, GitLab, Bitbucket, TFS, VSTS.

Each Pull Request that SeaLights has UCC data on is annotated with the number of UCCs in the Pull Request as displayed in *Figure 4: Identifying Untested Code Changes with the SeaLights Chrome Extension (Pull Request list)* above.

A full list of changed files from all commits, up to and including the latest commit, is displayed in the Files Changed tab for each individual Pull Request as shown below.



[SLDEV-10565] show alert on upload empty csv #568

Yuval-Ben-Arie wants to merge 6 commits into `master` from `10565-show-alert-on-upload-empty-csv`

9% New/Modified Code Coverage

5% Overall Code Coverage

5012 New/Modified & Untested Code Elements

src/app/core/models/alert-dialog.model.ts

```
@@ -10,4 +10,5 @@ export interface AlertDialogData {
    showInput?: boolean;
    inputValue?: string;
    inputLabel?: string;
+   closeOnConfirm?: boolean;
  }
```

src/app/shared/dialog/components/alert-dialog/alert-dialog.component.html

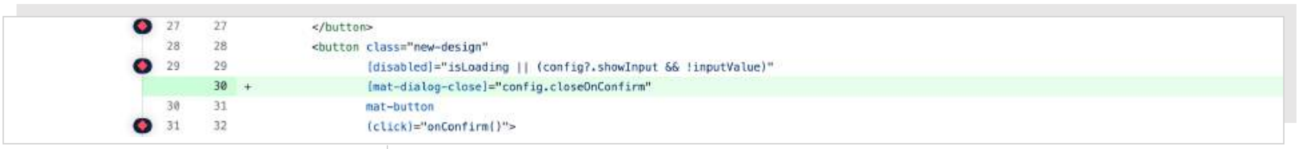
```
<sl-dialog [title]="config.title">
    <ng-container content>
        <div class="sl-flexbox-column sl-flexbox-gap-16">
            <p [innerHTML]="config.message"></p>
            <div *ngIf="config.showInput" class="sl-flexbox-column sl-flexbox-gap-8">
                <label *ngIf="config.inputLabel" class="input-label uppercase sl-typography-label" for="input-value">{{config.inputLabel}}</label>
                <input id="input-value"
                        pInputText
                        type="text"
                        placeholder=""
                        (input)="onInputChange()"
                        [(ngModel)]="inputValue"
                        autocorrect="off"
                        spellcheck="false"
                        autocomplete="off">
                <span *ngIf="error" class="sl-typography-body-medium error">{{ error }}</span>
            </div>
        </div>
    </ng-container>
    <ng-container actions>
        <button *ngIf="config.cancelButtonCaption?.length"
                [mat-dialog-close]="true"
                class="sl-typography-link-action"
                mat-button
                [disabled]="isLoading">
            {{ config.cancelButtonCaption }}
        </button>
        <button class="new-design"
                [disabled]="isLoading || (config?.showInput && !inputValue)"
+               [mat-dialog-close]="config.closeOnConfirm"
                mat-button
                (click)="onConfirm()">
            {{ config.confirmButtonCaption }}
        </button>
    </ng-container>
</sl-dialog>
```

Figure 5: Identifying Untested Code Changes with the SeaLights Chrome Extension (Individual Pull Request)

The metrics reflect the selected test stage and the coverage type which can be:

- A single coverage type (method or branch)
- All coverage types (if the repository provides metrics of multiple coverage types)

Each line of code that is identified as containing an UCC will be annotated by a red diamond as shown below.



```
      </button>
      <button class="new-design"
              [disabled]="isLoading || (config?.showInput && !inputValue)"
+             [mat-dialog-close]="config.closeOnConfirm"
              mat-button
              (click)="onConfirm()">
```

Figure 6: Untested Code Change annotation

CODE REVIEW AND APPROVAL BEST PRACTICE

Code Reviewer functionality provided by the SeaLights Chrome Extension is typically used by the code reviewer or managers to review and comment on changes prior to approving the merge of the pull request. The review should be a collaborative process involving the software development and quality engineering teams who will look at the UCCs identified by SeaLights and then decide if the impact of the UCCs meet the repository's contributing guidelines and other quality standards.

To decide whether to approve a pull request you should review the results within the Pull Request view in your SCM, as shown in *Figure 4: Identifying Untested Code Changes with the SeaLights Chrome Extension (Pull Request list)*. This will highlight the number of UCCs per Pull Request.

Selecting a specific Pull Request and selecting the **Files Changed** tab will enable you to see the UCCs inline in your SCM, denoted by a red diamond as shown in *Figure 6: Untested Code Change annotation*. This is where the software development and quality engineering teams collaborate to decide on whether the code is of suitable quality to be merged.

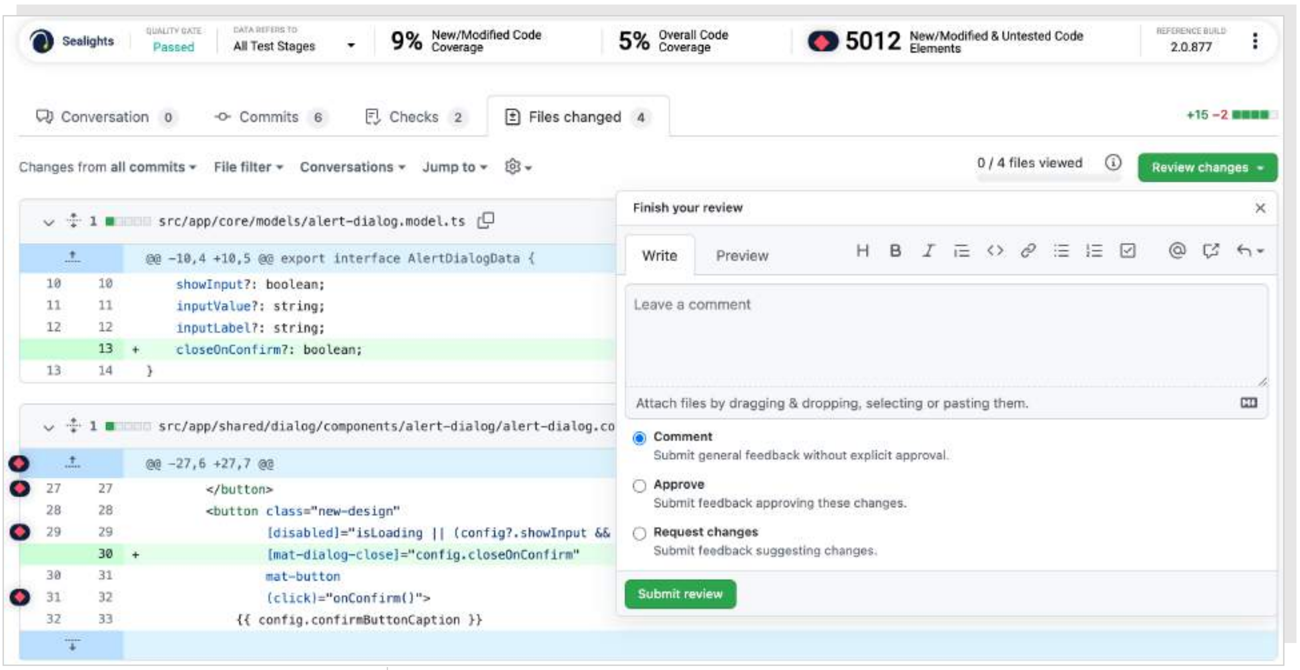
After a pull request is opened, anyone with read access can review and comment on the changes it proposes. You can also suggest specific changes to lines of code, which the author can apply directly from the pull request.

Repository owners and collaborators can request a pull request review from a specific person. Organization members can also request a pull request review from a team with read access to the repository.

Reviews allow for discussion of proposed changes and help ensure that the changes meet the repository's contributing guidelines and other quality standards. You can define which individuals or teams own certain types or areas of code in a CODEOWNERS file. When a pull request modifies code that has a defined owner, that individual or team will automatically be requested as a reviewer.

A review has three possible statuses:

- **Comment:** Submit general feedback without explicitly approving the changes or requesting additional changes.
- **Approve:** Submit feedback and approve merging the changes proposed in the pull request.
- **Request changes:** Submit feedback that must be addressed before the pull request can be merged.



```
src/app/core/models/alert-dialog.model.ts
@@ -10,4 +10,5 @@ export interface AlertDialogData {
      showInput?: boolean;
      inputValue?: string;
      inputLabel?: string;
+     closeOnConfirm?: boolean;
  }
```

```
src/app/shared/dialog/components/alert-dialog/alert-dialog.co
@@ -27,6 +27,7 @@
          </button>
          <button class="new-design"
                  [disabled]="isLoading || (config?.showInput &&
+                 [mat-dialog-close]="config.closeOnConfirm"
                  mat-button
                  (click)="onConfirm()">
              {{ config.confirmButtonCaption }}
```

[✓ Comment] Submit general feedback without explicit approval.
Approve Submit feedback approving these changes.
Request changes Submit feedback suggesting changes.

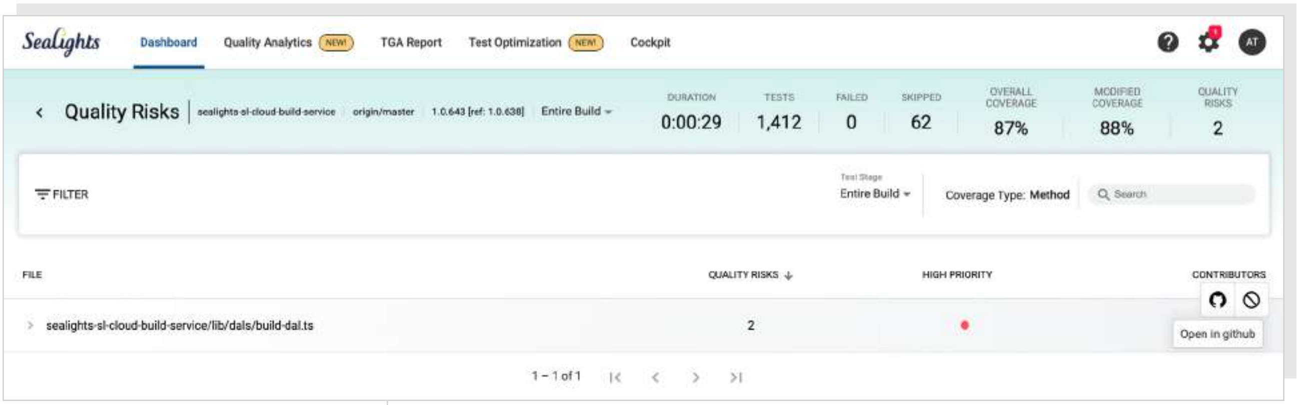
Figure 7: Example of setting Pull Request Review Status in GitHub

ANALYZING QUALITY RISKS

The SeaLights Chrome Extension is also used to present UCCs inline via the **Quality Risks** reported in the SeaLights Dashboard as shown in *Figure 2: Quality Risks*.

In addition to providing the name of the contributor and any commit message you can also identify the file name, line number and method where the quality risk has been identified.

Clicking on the logo that represents the SCM will take you directly to the line of code in your repository as shown in *Figure 8: Analyzing Quality Risks (Navigating to your SCM)* and *Figure 9: Analyzing Quality Risks (SCM Integration).*



Quality Risks | sealights-sl-cloud-build-service | origin/master | 1.0.643 [ref: 1.0.638] | Entire Build

DURATION	TESTS	FAILED	SKIPPED	OVERALL COVERAGE	MODIFIED COVERAGE	QUALITY RISKS
0:00:29	1,412	0	62	87%	88%	2

Test Stage: Entire Build | Coverage Type: Method

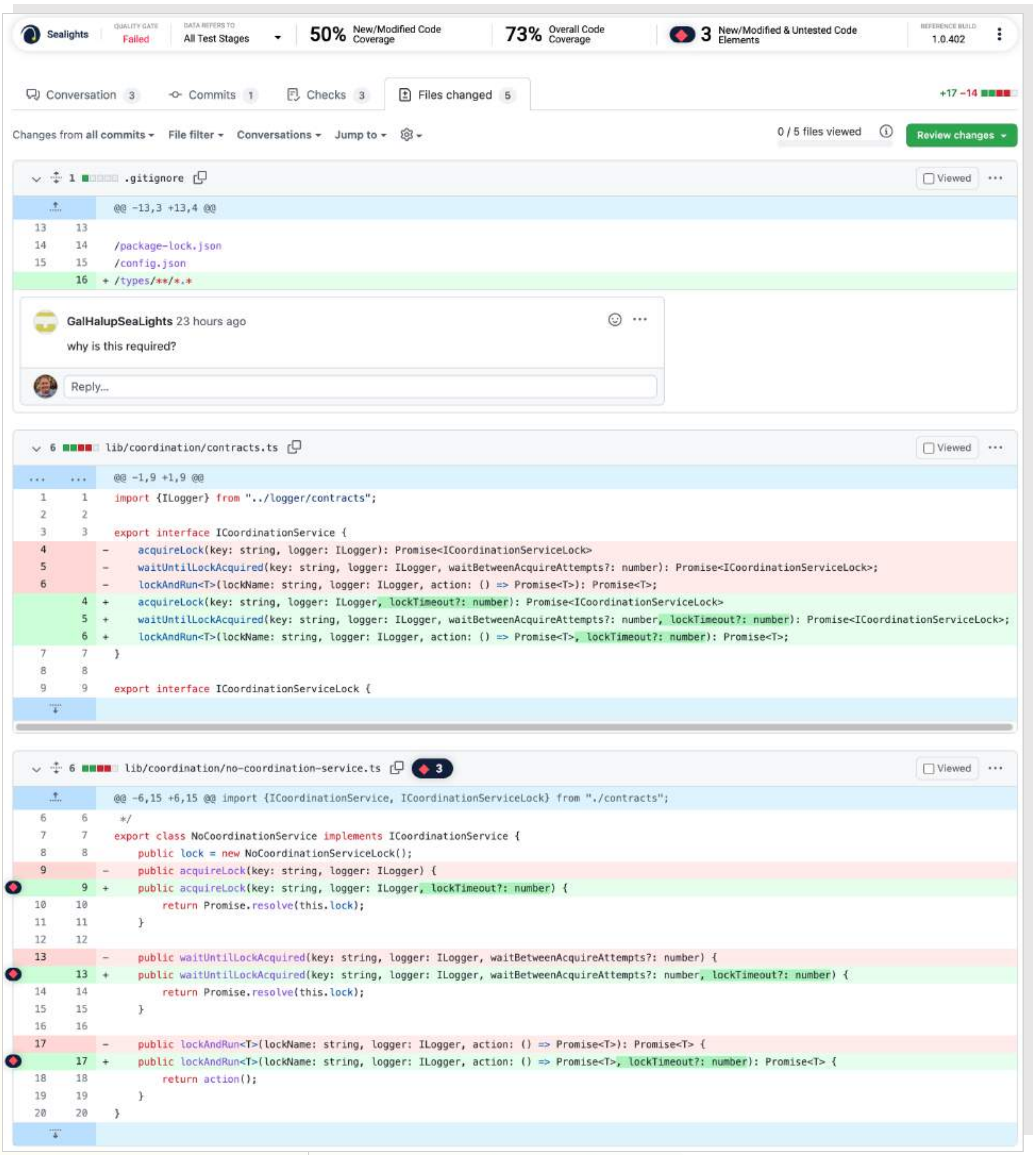
FILE	QUALITY RISKS	HIGH PRIORITY	CONTRIBUTORS
sealights-sl-cloud-build-service/lib/dals/build-dal.ts	2	●	

Figure 8: Analyzing Quality Risks (Navigating to your SCM)

ANALYZING COVERAGE DATA PROCESS OVERVIEW

In this phase of the overall process, we have reviewed the best practice for using the SeaLights functionality to establish a consistent methodology to perform a deep analysis of code coverage and potential quality risks. To summarise:

1. Assess the impact of Quality Risks highlighted by SeaLights in a code review.
2. Identify the contributors who can help build coverage for the Quality Risks.
3. Have the software development and quality engineering teams collaborate in defining a plan to address all Quality Risks.



1 .gitignore

```
@@ -13,3 +13,4 @@

  /package-lock.json
  /config.json
+ /types/**/*.*
```

GalHalupSeaLights 23 hours ago

why is this required?

6 lib/coordination/contracts.ts

```
@@ -1,9 +1,9 @@
  import {ILogger} from "../logger/contracts";

  export interface ICoordinationService {
-     acquireLock(key: string, logger: ILogger): Promise<ICoordinationServiceLock>
-     waitUntilLockAcquired(key: string, logger: ILogger, waitBetweenAcquireAttempts?: number): Promise<ICoordinationServiceLock>;
-     lockAndRun<T>(lockName: string, logger: ILogger, action: () => Promise<T>): Promise<T>;
+     acquireLock(key: string, logger: ILogger, lockTimeout?: number): Promise<ICoordinationServiceLock>
+     waitUntilLockAcquired(key: string, logger: ILogger, waitBetweenAcquireAttempts?: number, lockTimeout?: number): Promise<ICoordinationServiceLock>;
+     lockAndRun<T>(lockName: string, logger: ILogger, action: () => Promise<T>, lockTimeout?: number): Promise<T>;
  }

  export interface ICoordinationServiceLock {
```

6 lib/coordination/no-coordination-service.ts 3

```
@@ -6,15 +6,15 @@ import {ICoordinationService, ICoordinationServiceLock} from "./contracts";
   */
  export class NoCoordinationService implements ICoordinationService {
      public lock = new NoCoordinationServiceLock();
-     public acquireLock(key: string, logger: ILogger) {
+     public acquireLock(key: string, logger: ILogger, lockTimeout?: number) {
          return Promise.resolve(this.lock);
      }

-     public waitUntilLockAcquired(key: string, logger: ILogger, waitBetweenAcquireAttempts?: number) {
+     public waitUntilLockAcquired(key: string, logger: ILogger, waitBetweenAcquireAttempts?: number, lockTimeout?: number) {
          return Promise.resolve(this.lock);
      }

-     public lockAndRun<T>(lockName: string, logger: ILogger, action: () => Promise<T>): Promise<T> {
+     public lockAndRun<T>(lockName: string, logger: ILogger, action: () => Promise<T>, lockTimeout?: number): Promise<T> {
          return action();
      }
  }
```

Figure 9: Analyzing Quality Risks (SCM Integration)

STEP 3 – DEFINE QUALITY POLICIES

3

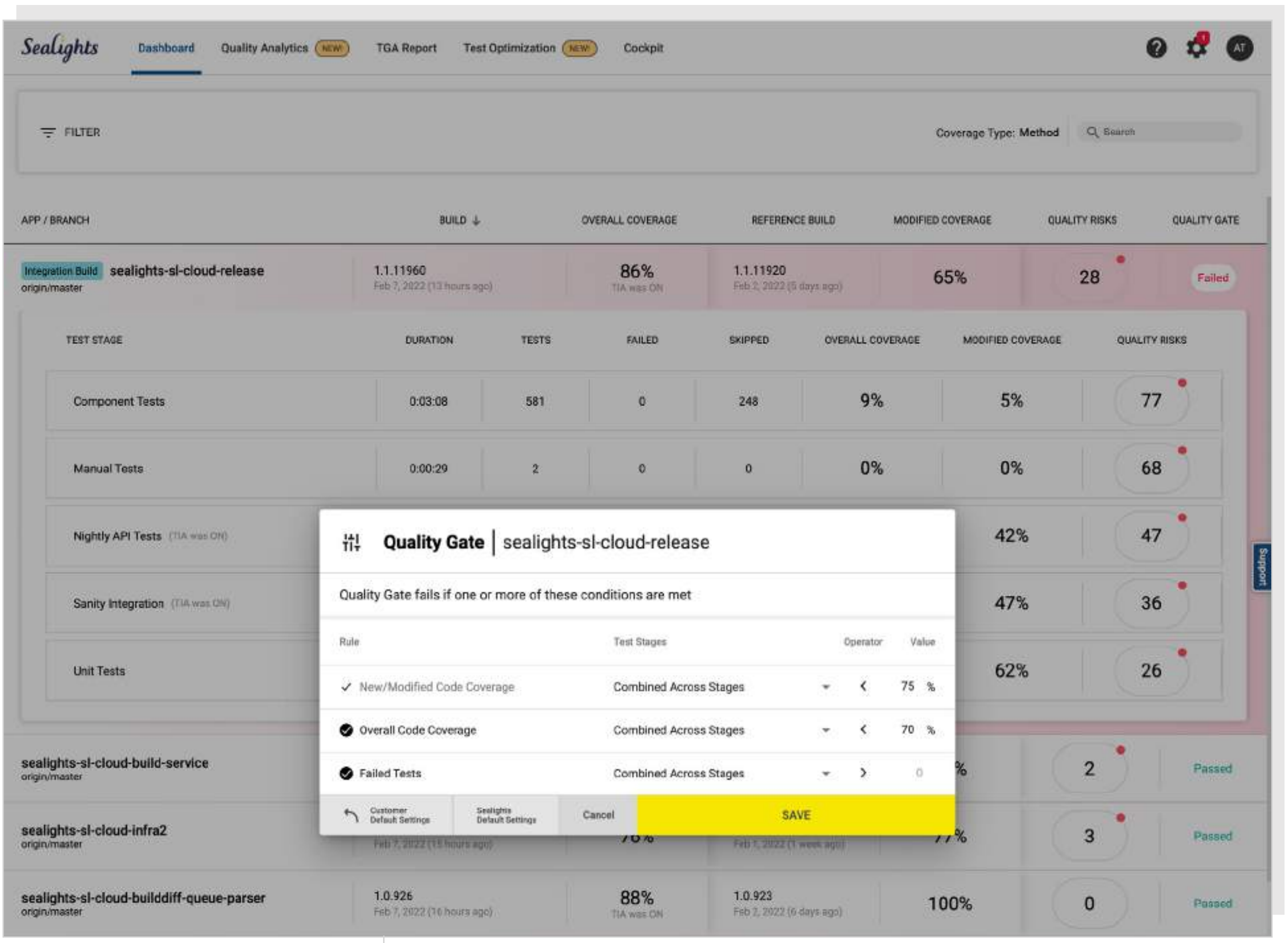
The third step, or "Line of Defence", in implementing a Software Quality Intelligence process involves defining **Quality Policies** based on risk levels and automatic gates to accelerate processes.

PERSONAS AND CADENCE

Typically the best practices that will be developed and implemented in this step would be led by, but not be limited to, the Quality Engineering organization based on feedback from the product owners on the business criticality/impact of the code. Quality Gates should then be regularly reviewed to determine if the criteria can be increased to drive a quality culture based upon continuous improvement.

FUNCTIONAL OVERVIEW

Before we review the process let's look at the SeaLights functionality that enables this.



APP / BRANCH	BUILD ↓	OVERALL COVERAGE	REFERENCE BUILD	MODIFIED COVERAGE	QUALITY RISKS	QUALITY GATE
Integration Build sealights-sl-cloud-release origin/master	1.1.11960 Feb 7, 2022 (13 hours ago)	86% TIA was ON	1.1.11920 Feb 2, 2022 (5 days ago)	65%	28	Failed
sealights-sl-cloud-build-service origin/master				%	2	Passed
sealights-sl-cloud-infra2 origin/master	Feb 7, 2022 (15 hours ago)	76%	Feb 1, 2022 (1 week ago)	77%	3	Passed
sealights-sl-cloud-builddiff-queue-parser origin/master	1.0.926 Feb 7, 2022 (16 hours ago)	88% TIA was ON	1.0.923 Feb 2, 2022 (6 days ago)	100%	0	Passed

TEST STAGE	DURATION	TESTS	FAILED	SKIPPED	OVERALL COVERAGE	MODIFIED COVERAGE	QUALITY RISKS
Component Tests	0:03:08	581	0	248	9%	5%	77
Manual Tests	0:00:29	2	0	0	0%	0%	68
Nightly API Tests (TIA was ON)						42%	47
Sanity Integration (TIA was ON)						47%	36
Unit Tests						62%	26

Quality Gate | sealights-sl-cloud-release

Quality Gate fails if one or more of these conditions are met

Rule	Test Stages	Operator	Value
✓ New/Modified Code Coverage	Combined Across Stages	<	75 %
✓ Overall Code Coverage	Combined Across Stages	<	70 %
✓ Failed Tests	Combined Across Stages	>	0

Customer Default Settings | Sealights Default Settings | Cancel | SAVE

Figure 10: Defining Quality Gate Criteria

The SeaLights Quality Gate defines thresholds that determine if a build is meeting your quality criteria and can be set differently for each application/service, or applied consistently across all applications/services.

There are three criteria that control the Quality Gate status:

- New/Modified Code Coverage
- Overall Code Coverage
- Failed Tests

Quality Gate status can be **Passed** (all the quality gates have met all the criteria) or **Failed** (one, or more, of the quality gates did not meet the criteria). SeaLights provides the ability to generate Slack and Email notifications on builds reported to the Dashboard. Quality Gate status is a trigger to generate notifications.

BEST PRACTICES FOR DEFINING QUALITY GATE CRITERIA – COVERAGE GOALS

Quality Gates are part of the software development process that use specific, measurable, and achievable criteria for each build of each individual component. Enabling Quality Gates enforces the improvement of each development stage, making the process of elevating quality more transparent and traceable whilst maintaining, and increasing, velocity by providing data that enables real-time decision making and ultimately automates the promotion of code.

Focusing on getting 100% code coverage can on its own create technical debt from low value tests and additional maintenance effort. You should also pay attention to tests that have been copies and pasted just to seemingly increase coverage to hit the target coverage goal.

However, a low code coverage number and the resulting high number of quality risks increases the likelihood of pushing bad code into production. SeaLights focuses on highlighting not what code is covered, but what's not covered.

There is not one ideal coverage goal that you can universally apply to all applications/services. The level of coverage should be a function of:

- Business impact/criticality of the code
- How often the code will be modified
- Lifespan of the code
- Code complexity

Ultimately, the level of code coverage is a business decision made by the product owners with the domain knowledge. However, any mandated target goal should be supported by infrastructure investments to ensure that the correct tooling is made available to develop, test and govern the process. The most effective way of setting a target goal is typically to have the team select the value that makes the sense for their business need.

As the level of code coverage increases the gains in quality diminish so the focus should be on significant returns, for example getting from 30% to 70% and maintaining that level of coverage. Applying the processes outlined in this document, combined with increased visibility, will automatically raise code coverage levels well beyond target goals. For example, collaborating on UCCs that SeaLights has identified, that take place during the code review process, are more valuable than simply focusing on a coverage goal figure. Embedding code coverage into your code review process will make code reviews faster, easier, and prioritised based on Quality Risk.

STEP 4 – PERFORM A RETROSPECTIVE ANALYSIS

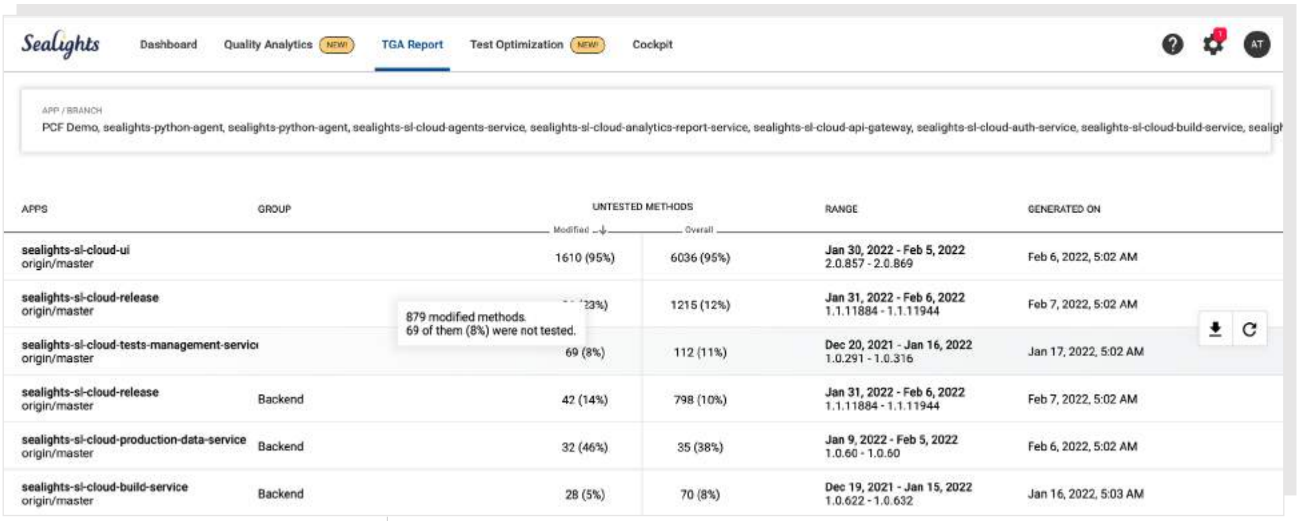
In the "Report" phase of implementing a Software Quality Intelligence process we provide the ability to produce sprint retrospectives to aid in the planning process for subsequent sprints with **Test Gap Analytics (TGA) Reports**.

PERSONAS AND CADENCE

Retrospective analysis is a collaborative exercise led by the Manager of the sprint. Regular test gap analysis can help managers improve test planning, keep the primary focus on testing new code, and also ensure good test coverage. SeaLights Test Gap Analytics gives clear visibility into the quality risks that accumulate over time.

FUNCTIONAL OVERVIEW

Before we review the process let’s look at the SeaLights functionality that enables this.



APP / BRANCH
PCF Demo, sealights-python-agent, sealights-python-agent, sealights-sl-cloud-agents-service, sealights-sl-cloud-analytics-report-service, sealights-sl-cloud-api-gateway, sealights-sl-cloud-auth-service, sealights-sl-cloud-build-service, sealigh

APPS	GROUP	UNTESTED METHODS Modified	UNTESTED METHODS Overall	RANGE	GENERATED ON
sealights-sl-cloud-ui origin/master		1610 (95%)	6036 (95%)	**Jan 30, 2022 - Feb 5, 2022** 2.0.857 - 2.0.869	Feb 6, 2022, 5:02 AM
sealights-sl-cloud-release origin/master		[illegible]23%)	1215 (12%)	**Jan 31, 2022 - Feb 6, 2022** 1.1.11884 - 1.1.11944	Feb 7, 2022, 5:02 AM
sealights-sl-cloud-tests-management-servic origin/master		69 (8%)	112 (11%)	**Dec 20, 2021 - Jan 16, 2022** 1.0.291 - 1.0.316	Jan 17, 2022, 5:02 AM
sealights-sl-cloud-release origin/master	Backend	42 (14%)	798 (10%)	**Jan 31, 2022 - Feb 6, 2022** 1.1.11884 - 1.1.11944	Feb 7, 2022, 5:02 AM
sealights-sl-cloud-production-data-service origin/master	Backend	32 (46%)	35 (38%)	**Jan 9, 2022 - Feb 5, 2022** 1.0.60 - 1.0.60	Feb 6, 2022, 5:02 AM
sealights-sl-cloud-build-service origin/master	Backend	28 (5%)	70 (8%)	**Dec 19, 2021 - Jan 15, 2022** 1.0.622 - 1.0.632	Jan 16, 2022, 5:03 AM

879 modified methods.
69 of them (8%) were not tested.

Figure 11: SeaLights TGA Report

Test Gap Analytics identifies all quality risks for a specific time period, and includes the following:

- all builds
- all test stages
- all code changes

SPRINT RETROSPECTIVES AND PLANNING WITH TEST GAP ANALYTICS (TGA) REPORTS

Before you start using TGA Reports you should set the scope of the code that matters to you. You do this by removing data that you determine will be irrelevant, steps to do this can be found in the product documentation on “Settings Area”.

Having defined the data in scope, you can then apply three use cases:

- Definition of Done for sprint quality - Validate new/modified code has been tested
- Monthly Quality Report - Analyze the quality performance of your applications/teams
- Test Development - Identify high-risk code areas and create testing plan

We recommend that the Monthly Quality Report should be used as the Quality KPI going forward.

Use these settings to define the Test Gaps Analytics reports - setting the reports here will result in reports that will be available in the Test Gaps Analytics screen.

ADDING AN APPLICATION TO THE TEST GAP ANALYTICS

1. Select the relevant application
2. Select the relevant branch
3. Select the start date (a future date, e.g. next Sprint start date or the 1st of the next Month)
4. Select the reporting period in weeks (Scheduled refresh)
5. Click “Save”

In the example shown in *Figure 11: SeaLights TGA Report Detailed Test Gap Analysis* you can see how we can look back at a specific date range to get a view of what gaps remain that may impact quality, and the specific details of which methods may introduce defects.

If you are using the SeaLights **Production Listener** you will have an additional two columns representing Methods **Used in Production** and **Modified and Used in Production**.

Drilling down, by selecting the application/service, and sorting/filtering enables you to quickly access the details of specific areas of your code which will help you identify the areas of code to focus on.

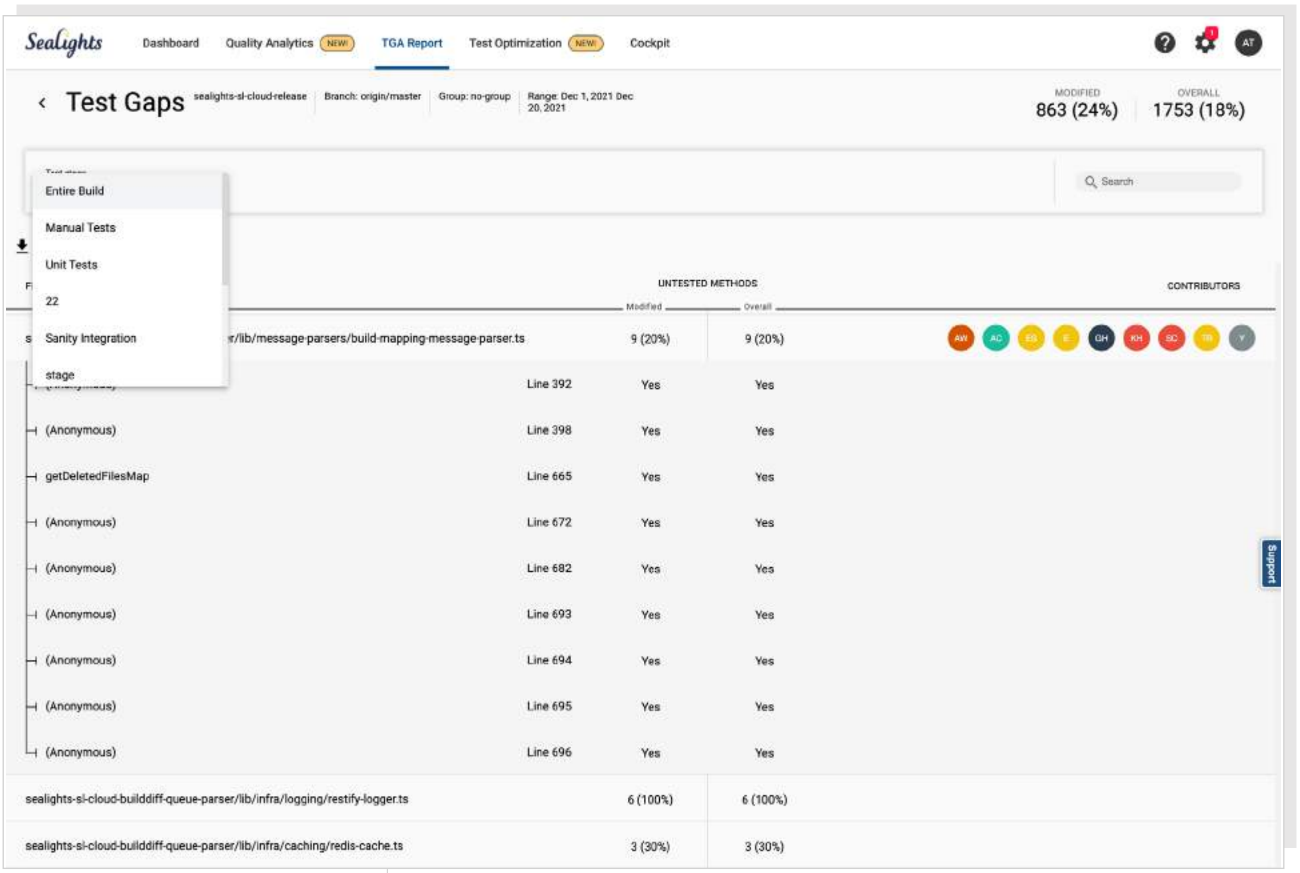
a. Use the **"Search"** bar to focus on important classes/files (Logic, Calculation, etc…)

b. Use the **"Test Stage"** drop-down button to review the untested files/methods per specific test stage. Focus on areas which are sensitive to Integration Tests:

   i. Priority 0 - Untested by all test stages

   ii. Priority 1 - Untested by a specific test stage, i.e. Integration/Automation/Regression Tests (the rationale of this is to exclude the unit tests)

c. For **legacy applications** (**low code change volume**), focus on the modified areas



Test Gaps sealights-sl-cloud-release | Branch: origin/master | Group: no-group | Range: Dec 1, 2021 Dec 20, 2021

MODIFIED 863 (24%)   OVERALL 1753 (18%)

		UNTESTED METHODS Modified	Overall	CONTRIBUTORS
…r/lib/message-parsers/build-mapping-message-parser.ts		9 (20%)	9 (20%)	AW AC SS E GH KH SC TR Y
	Line 392	Yes	Yes	
(Anonymous)	Line 398	Yes	Yes	
getDeletedFilesMap	Line 665	Yes	Yes	
(Anonymous)	Line 672	Yes	Yes	
(Anonymous)	Line 682	Yes	Yes	
(Anonymous)	Line 693	Yes	Yes	
(Anonymous)	Line 694	Yes	Yes	
(Anonymous)	Line 695	Yes	Yes	
(Anonymous)	Line 696	Yes	Yes	
sealights-sl-cloud-builddiff-queue-parser/lib/infra/logging/restify-logger.ts		6 (100%)	6 (100%)	
sealights-sl-cloud-builddiff-queue-parser/lib/infra/caching/redis-cache.ts		3 (30%)	3 (30%)	

Figure 12: SeaLights TGA Report Detailed Test Gap Analysis

If you have the SeaLights Chrome Extension installed, you can access more detail in your SCM.

1. Click on the SCM link for the given file you would want to explore further
2. You will be linked to your code repository and can access the specified file space within your SCM.
    a. You can see the TGA insights in line with your code with the SeaLights "Code Viewer", part of SeaLights Chrome Extension
    b. SCM's supported by SeaLights Code Viewer– Github, BitBucket, Gitlab, TFS/TFVC
3. Use the test stage drop-down button to focus on a specific test stage
    a. e.g. In order to focus on areas which are not tested by Manual Tests > pick the "Manual Tests" value from the drop-down
4. Review the file and Quality Risks and decide which of these should be addressed within the current/upcoming Sprint
5. Control and Feedback Loop
    a. Reiterate the process in future sprints to ensure that the gaps have been actually closed by your Dev/QA team and identify if new test gaps have been created
    b. Create a monthly report which includes the relevant sprint

STEP 5 – ASSESS OVERALL QUALITY TRENDS

5

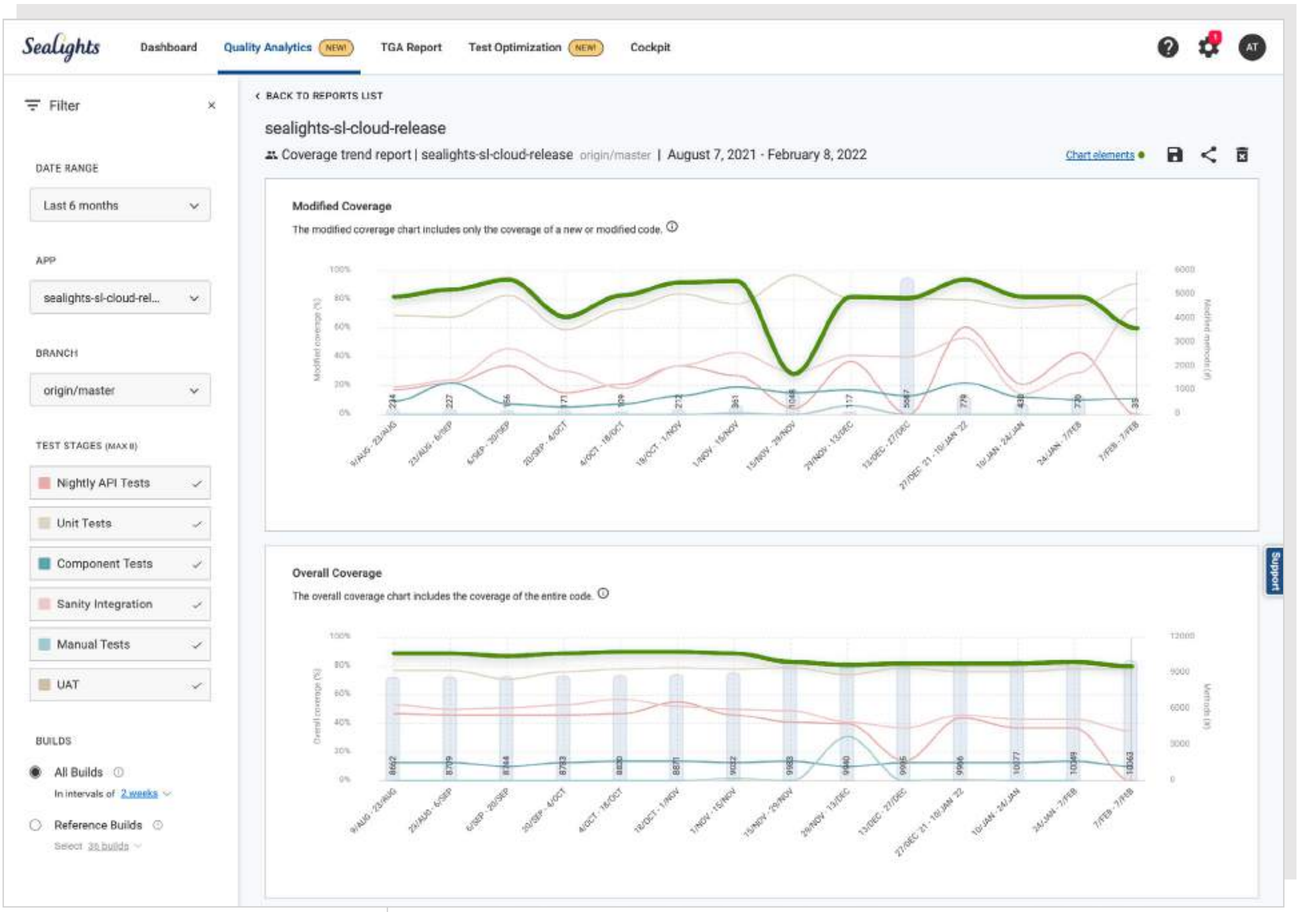
In the "Trend Reporting" phase of implementing a Software Quality Intelligence process we take a high level look at how quality risks associated with application code have developed over time with the SeaLights **Quality Analytics Coverage Trend Reports**.

PERSONAS AND CADENCE

Quality Analytics Coverage Trends Reports are focused on providing senior management business level visibility into how quality is developing over time and the impact of different testing strategies. We recommend that this should be reviewed monthly at a minimum.

FUNCTIONAL OVERVIEW

Before we review the process let's look at the SeaLights functionality that enables this.



sealights-sl-cloud-release

Coverage trend report | sealights-sl-cloud-release origin/master | August 7, 2021 - February 8, 2022

The modified coverage chart includes only the coverage of a new or modified code.

The overall coverage chart includes the coverage of the entire code.

DATE RANGE: Last 6 months

APP: sealights-sl-cloud-rel...

BRANCH: origin/master

TEST STAGES (MAX 8):
- Nightly API Tests ✓
- Unit Tests ✓
- Component Tests ✓
- Sanity Integration ✓
- Manual Tests ✓
- UAT ✓

BUILDS:
- ● All Builds — In intervals of 2 weeks
- ○ Reference Builds — Select 38 builds

Figure 13: SeaLights Quality Analytics – Coverage Trend Report

You can analyze what affects your quality the most, what is the impact of your different test stages, how your quality trend looks like, where your quality pain points are and more.

The **Coverage Trend Report** is the first report available under Quality Analytics. It displays application code coverage over time.

When you first enter the Quality Analytics page, you will see an empty list of saved reports. Click on the + button at the right top corner to create your first report.

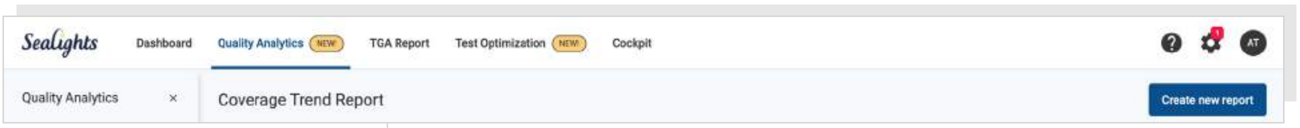


Figure 14: Creating a Coverage Trend Report

Once you create a report you have to click the save button, to add it to your saved report list. That way you can return to this report any time to track progress.

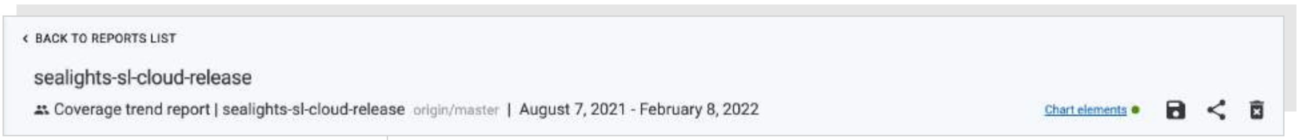

BACK TO REPORTS LIST
sealights-sl-cloud-release
Coverage trend report | sealights-sl-cloud-release origin/master | August 7, 2021 - February 8, 2022
Chart elements

Figure 15: Saving a Coverage Trend Report

The reports you create are private and only available for you. You can use the copy button to share a specific report, as described below:

Generating A Report

Generating a report is as simple as selecting the app and branch you are interested in.

You have different options to view the data:

Date Range

The popular options available are: Last month, Last 2 months, Last 3 months, Last 6 months, Last year.

If none of the above is what you need, you can always select a custom date range. We recommend a minimum of 4 weeks.

Test Stages

A list of all test stages reported for the selected app and branch during the past year, ordered alphabetically. Each test stage has its own color that is used in the charts.

You can select / deselect any of the test stages to focus on the data you want to see in the charts.

Builds

There are two ways to analyze quality data overtime: either look at all builds of the selected app/branch or only track builds marked as reference builds (usually these are production/released builds).

- When looking at all builds, you can select in which intervals you would like to aggregate the data. Options are: 1 week, 2 weeks, 3 weeks, 4 weeks, 1 month.
- When tracking only reference builds, you can select/deselect reference builds from a list (most likely you would like to include all reference builds).

Understanding The Data

The report includes two charts: Modified Coverage and Overall Coverage.

Both charts display an aggregated coverage on top of the specific test stages coverage and include:

- Y-axis: Coverage % (0-100%)
- X-axis: Time, according to the filter selection and intervals

Build Intervals

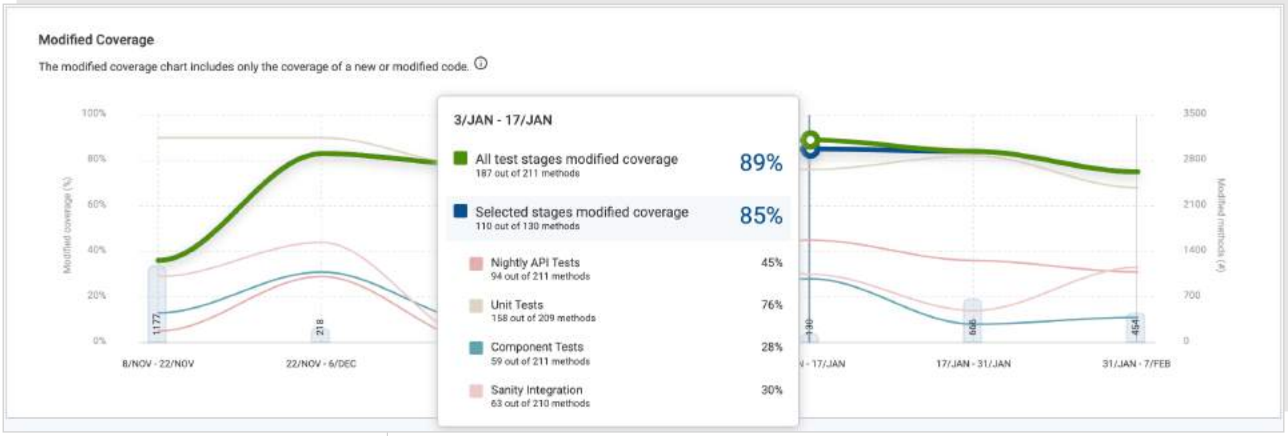
When selecting any of the interval options (1 week, 2 weeks, 3 weeks, 4 weeks, or 1 month) any point in the X-axis and in the chart line represents an interval. Interval X takes all code changes from the last build in interval X-1, and takes all coverage from all related test stages that were reported in all builds within interval X. (e.g. Interval of 1 month: coverage in July means all code changes since last build on June and their coverage from all test stages executions reported on builds within July).

Reference Builds

When selecting the reference build option any point in the X-axis and in the chart line represents a single reference build. Reference build X takes all code changes from reference build X-1, and takes only the coverage from the test stages that were reported to reference build X (similar to the calculation in the Dashboard).

Modified Coverage Chart

Modified coverage is the coverage of your code changes. Why is it important to track it? Your quality risks are code changes you haven't tested. The higher the coverage, the lower the chances for escaping defects. You want to see high coverage overtime, to reduce risks. High modified coverage overtime is an indication for a good quality culture. It means that any new or modified code is tested.



Modified Coverage

The modified coverage chart includes only the coverage of a new or modified code.

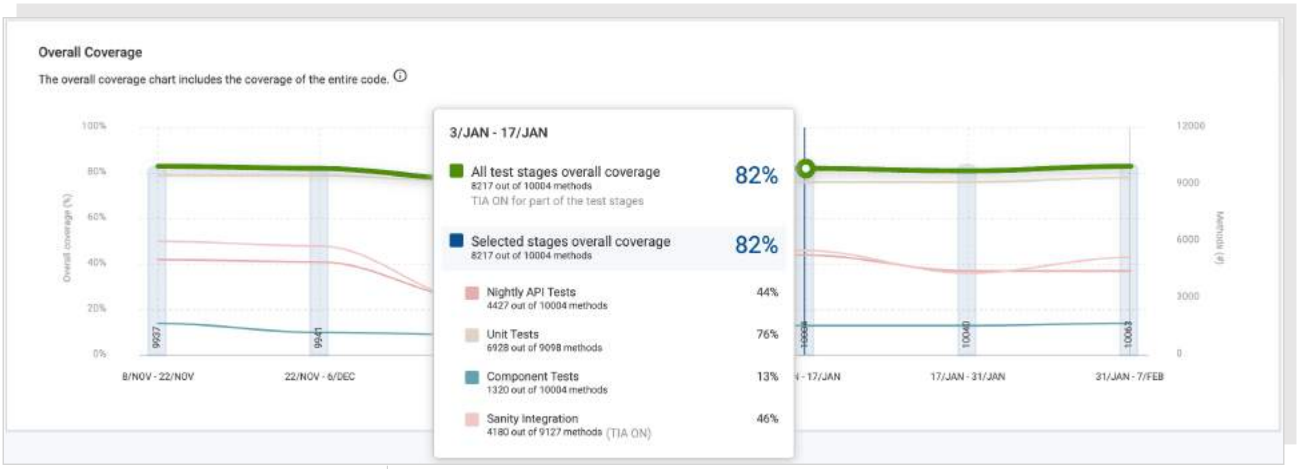
3/JAN - 17/JAN

🟩	All test stages modified coverage 187 out of 211 methods	89%
🟦	Selected stages modified coverage 110 out of 130 methods	85%
🟥	Nightly API Tests 94 out of 211 methods	45%
⬜	Unit Tests 158 out of 209 methods	76%
🟦	Component Tests 59 out of 211 methods	28%
🟥	Sanity Integration 63 out of 210 methods	30%

Figure 16: Coverage Trend Report (Modified Coverage)

Overall Coverage Chart

Overall coverage is the coverage of your entire code. You should track the overall coverage to make sure that overtime your trend is positive, mainly by covering new and modified code. You might see drops in cases you had major code changes with no coverage.



3/JAN - 17/JAN

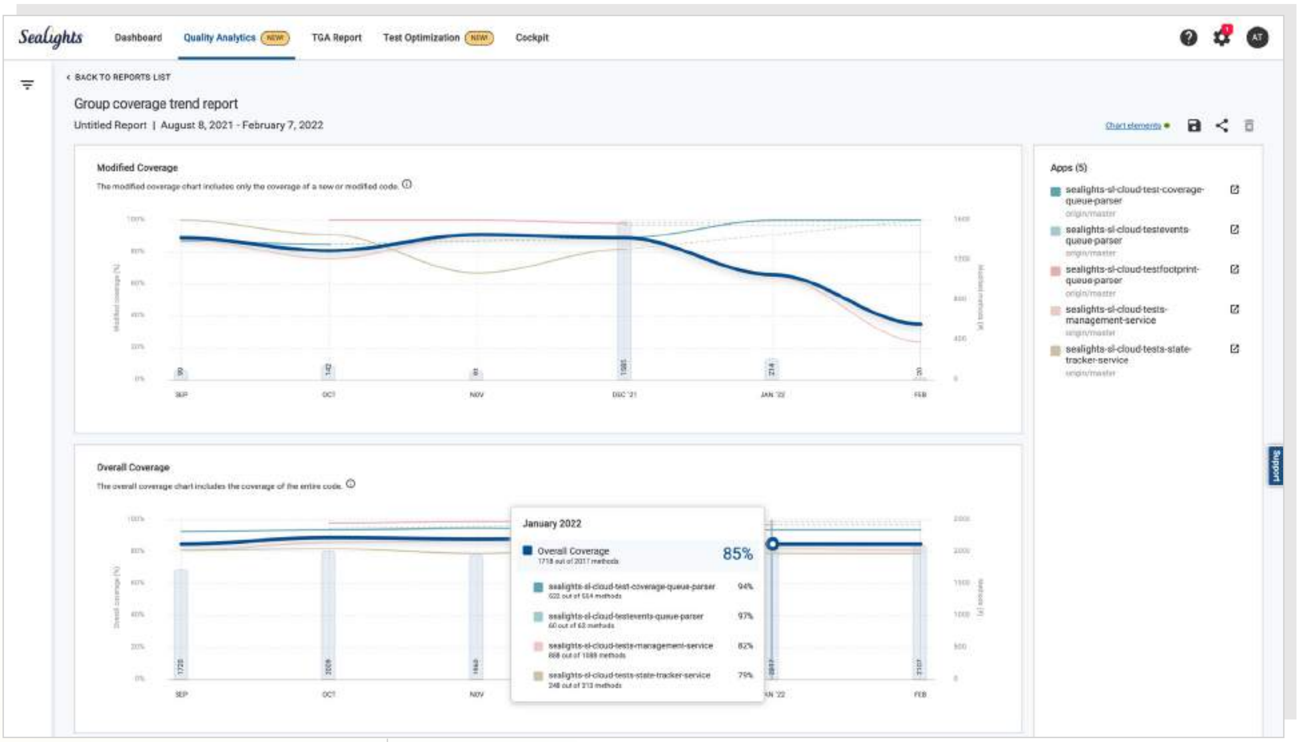
🟩	All test stages overall coverage 8217 out of 10004 methods TIA ON for part of the test stages	82%
🟦	Selected stages overall coverage 8217 out of 10004 methods	82%
🟥	Nightly API Tests 4427 out of 10004 methods	44%
⬜	Unit Tests 6928 out of 9098 methods	76%
🟦	Component Tests 1320 out of 10004 methods	13%
🟥	Sanity Integration 4180 out of 9127 methods (TIA ON)	46%

Figure 17: Coverage Trend Report (Overall Coverage)

Group Coverage Chart

Group coverage is the coverage for 2 or more applications (for a specific branch per application) providing a view of their individual coverage trend and an aggregated coverage trend (modified and overall coverage).

This report enables managers to track the coverage trend of an entire product, entire team, line of business, or a professional guild (for example Front-End, Java, Back-End, etc.).



Group coverage trend report

Untitled Report | August 8, 2021 - February 7, 2022

Modified Coverage

The modified coverage chart includes only the coverage of a new or modified code.

Overall Coverage

The overall coverage chart includes the coverage of the entire code.

January 2022

🟦 Overall Coverage 1718 out of 2017 methods	85%
sealights-sl-cloud-test-coverage-queue-parser 622 out of 664 methods	94%
sealights-sl-cloud-testevents-queue-parser 60 out of 62 methods	97%
sealights-sl-cloud-tests-management-service 888 out of 1088 methods	82%
sealights-sl-cloud-tests-state-tracker-service 248 out of 313 methods	79%

Apps (5)

- sealights-sl-cloud-test-coverage-queue-parser
  origin/master
- sealights-sl-cloud-testevents-queue-parser
  origin/master
- sealights-sl-cloud-testfootprint-queue-parser
  origin/master
- sealights-sl-cloud-tests-management-service
  origin/master
- sealights-sl-cloud-tests-state-tracker-service
  origin/master

Figure 18: Coverage Trend Report (Group Coverage)

Sharing The Report

After generating a report you can use the copy button to copy the report URL to your clipboard to share the report with others.

STEP 6 – OPTIMIZING EFFICIENCY/ VELOCITY

In the “Optimization” phase of implementing a Software Quality Intelligence process we use SeaLights’s capability to provide visibility into actual test execution time and test coverage requirements to reduce test cycles and improve test effectiveness.

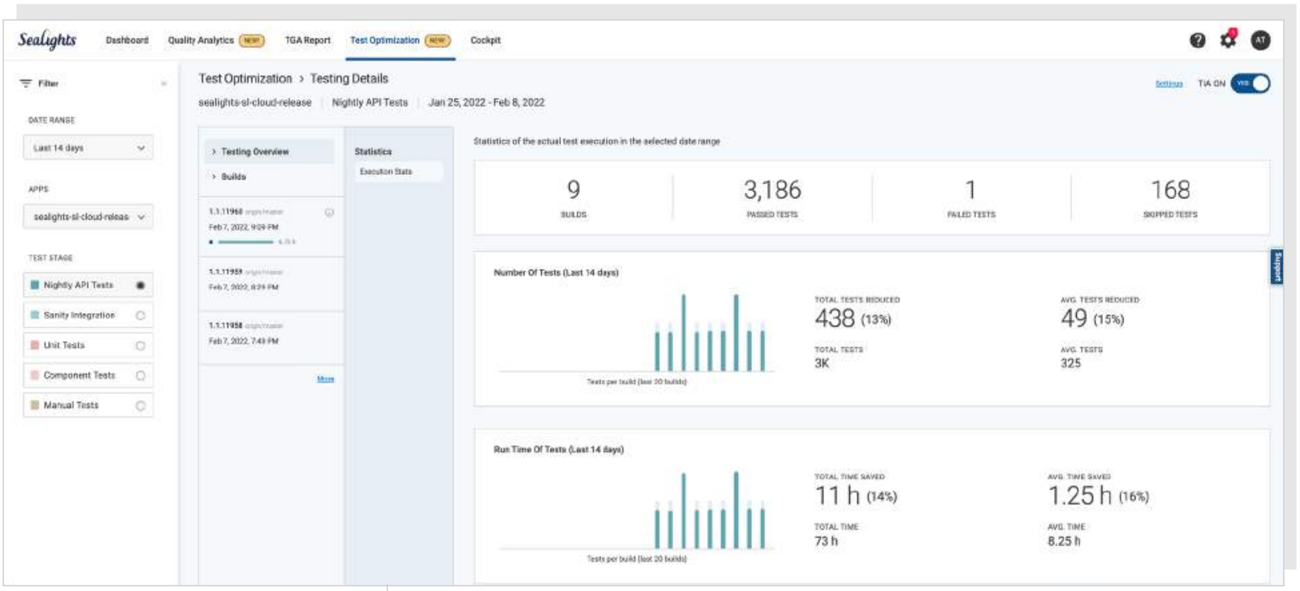
PERSONAS AND CADENCE

This step benefits all teams participating in the Software Development Lifecycle:

- Managers can schedule more test cycles by eliminating unnecessary tests
- Manual testers get test execution recommendations to increase their efficiency
- Developers get more code development iterations as testing cycle times are decreased

FUNCTIONAL OVERVIEW

Before we review the process let’s look at the SeaLights functionality that enables this.



Test Optimization › Testing Details

sealights-sl-cloud-release | Nightly API Tests | Jan 25, 2022 - Feb 8, 2022

DATE RANGE: Last 14 days

APPS: sealights-sl-cloud-releas

TEST STAGE: Nightly API Tests ● | Sanity Integration ○ | Unit Tests ○ | Component Tests ○ | Manual Tests ○

TIA ON

1.1.11960 — Feb 7, 2022, 9:09 PM

1.1.11959 — Feb 7, 2022, 8:29 PM

1.1.11958 — Feb 7, 2022, 7:49 PM

Statistics of the actual test execution in the selected date range

9	3,186	1	168
BUILDS	PASSED TESTS	FAILED TESTS	SKIPPED TESTS

Number Of Tests (Last 14 days)

TOTAL TESTS REDUCED 438 (13%) — TOTAL TESTS 3K

AVG. TESTS REDUCED 49 (15%) — AVG. TESTS 325

Run Time Of Tests (Last 14 days)

TOTAL TIME SAVED 11 h (14%) — TOTAL TIME 73 h

AVG. TIME SAVED 1.25 h (16%) — AVG. TIME 8.25 h

Figure 19: Test Optimization (Testing Details)

Test Optimization helps organizations shorten their automated and manual testing activities by reducing 50% to 90% of their test execution time. It is a Smart Test Execution Engine that cuts the testing cycle time by 50% to 90%. It can be tedious and time-consuming to run a full set of unit, functional, and regression tests each time a developer commits new code to the repository.

SeaLights Test Impact Analysis (TIA) eliminates much of this repetitive effort, since it can identify and execute the smallest subset of mandatory tests—without compromising quality.

Test Impact Analysis provides support for :

› all types of tests

› all of your apps and programming languages

SUPPORT FOR ALL TYPES OF TESTS

Through intelligent automation, SeaLights Test Impact Analysis will optimize and expedite the execution of almost any test type. Beside unit tests, you can apply TIA to your Continuous Integration (CI) implementation to automate functional, component, regression, and end-to-end tests. SeaLights TIA also accommodates manual testing and UI testing—including those using playback functionality (such as Selenium).

SUPPORT FOR ALL YOUR APPLICATIONS AND LANGUAGES

Besides integrating with your CI environment, SeaLights TIA supports various application architectures and programming languages. From monolithic applications to distributed systems, from HTTP methods to microservices, SeaLights TIA can accommodate most any architecture. There's no need to adapt your development practices to exploit TIA, since it supports Java, Node.js, JavaScript, and .NET/C#.

THE SEALIGHTS TEST IMPACT ANALYSIS PROCESS

TIA Outcome

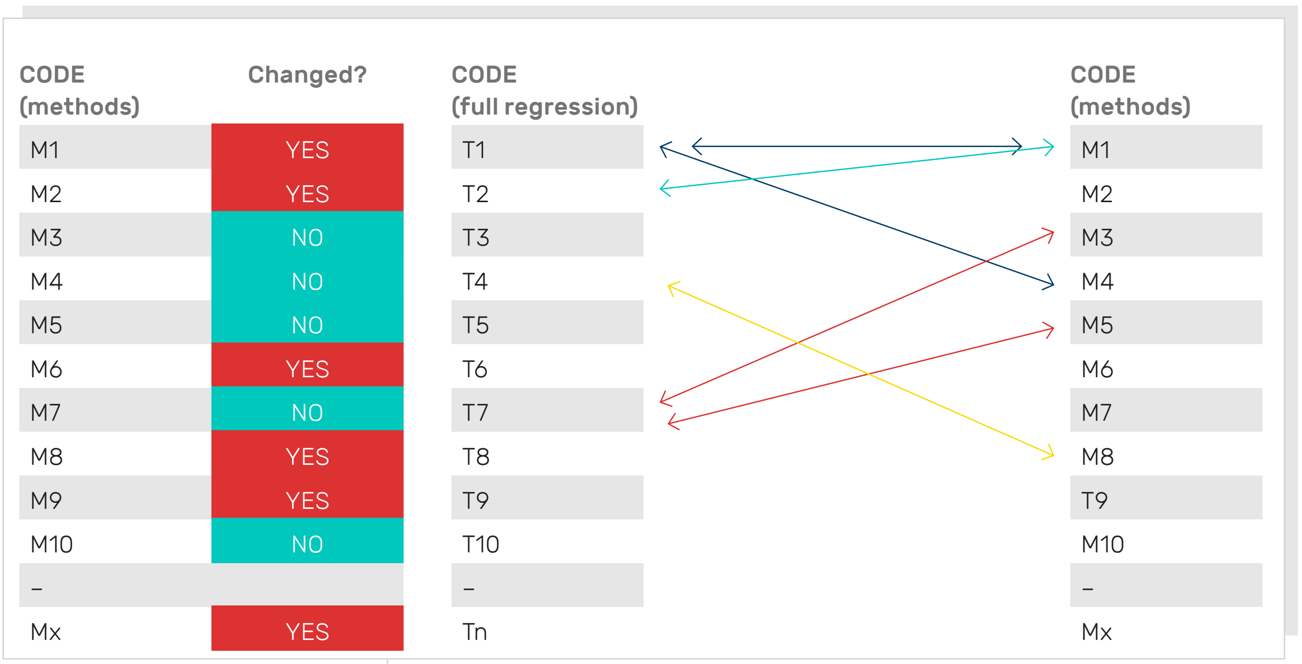
SeaLights Test Impact Analysis produces a list of test recommendations pertaining to these types of tests:

- Impacted tests - Tests that correspond to recent code changes.
- Failed Tests - Tests that did not pass in the previous run.
- Pinned/Flagged Test - Tests that are marked as important to run in every test cycle.
- New Tests - Tests that have never been run.

How does Test Impact Analysis (TIA) work?

These are the general steps in Test Impact Analysis:

1. You: Integrate SeaLights with your CI.
2. You: Install agents, then direct SeaLights to scan the latest build and the tests that are already running.
3. SeaLights: Correlate each test with the methods that relate to it.
4. SeaLights: Compare any new build content with the content with which the current test stage was last executed.
5. SeaLights: Identify any tests that are impacted by modified content and recommend tests.
6. You: Direct SeaLights to execute only the recommended tests, excluding those that are extraneous.
7. You: Execute "Full Run" tests on a defined schedule (for example, the first of day/week/month, or N builds) to ensure no tests are ever overlooked.
8. SeaLights: Analyze and improve using machine-learning algorithms.



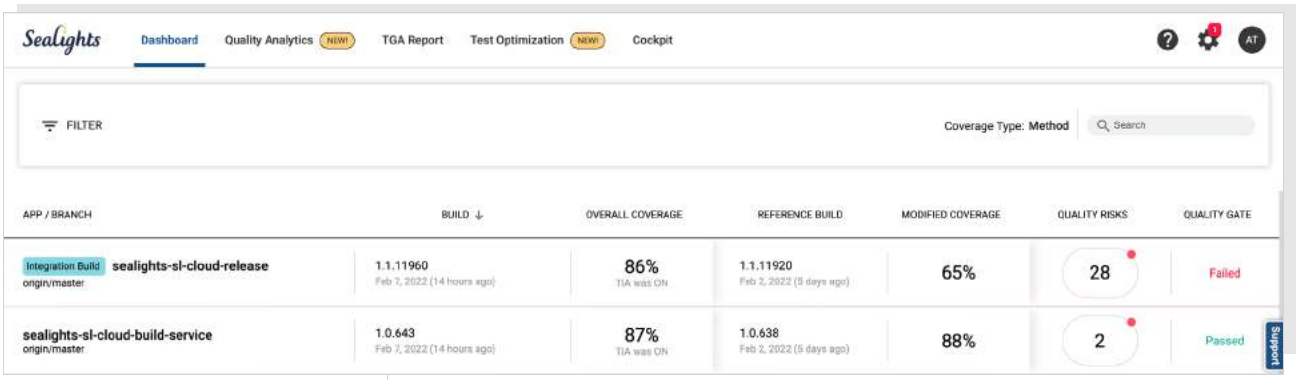
CODE (methods)	Changed?	CODE (full regression)	CODE (methods)
M1	YES	T1	M1
M2	YES	T2	M2
M3	NO	T3	M3
M4	NO	T4	M4
M5	NO	T5	M5
M6	YES	T6	M6
M7	NO	T7	M7
M8	YES	T8	M8
M9	YES	T9	T9
M10	NO	T10	M10
–		–	–
Mx	YES	Tn	Mx

Figure 20: How Test Optimization (Test Impact Analysis) Works

PRODUCT WALKTHROUGH

Follow these steps to configure TIA:

1. Click **Test Optimization** on the top right end of the screen.
2. The default configuration is that automatic test selection is “Off” (The TIA analysis will provide a list of test recommendations without executing the recommended test list).
3. Choose the **application name(s)** and the relevant branch(es).
4. Choose the **date range** of the TIA.
5. Select the **test stage(s)** to be analyzed



Coverage Type: Method

APP / BRANCH	BUILD ↓	OVERALL COVERAGE	REFERENCE BUILD	MODIFIED COVERAGE	QUALITY RISKS	QUALITY GATE
Integration Build sealights-sl-cloud-release origin/master	1.1.11960 Feb 7, 2022 (14 hours ago)	86% TIA was ON	1.1.11920 Feb 2, 2022 (5 days ago)	65%	28	Failed
sealights-sl-cloud-build-service origin/master	1.0.643 Feb 7, 2022 (14 hours ago)	87% TIA was ON	1.0.638 Feb 2, 2022 (5 days ago)	88%	2	Passed

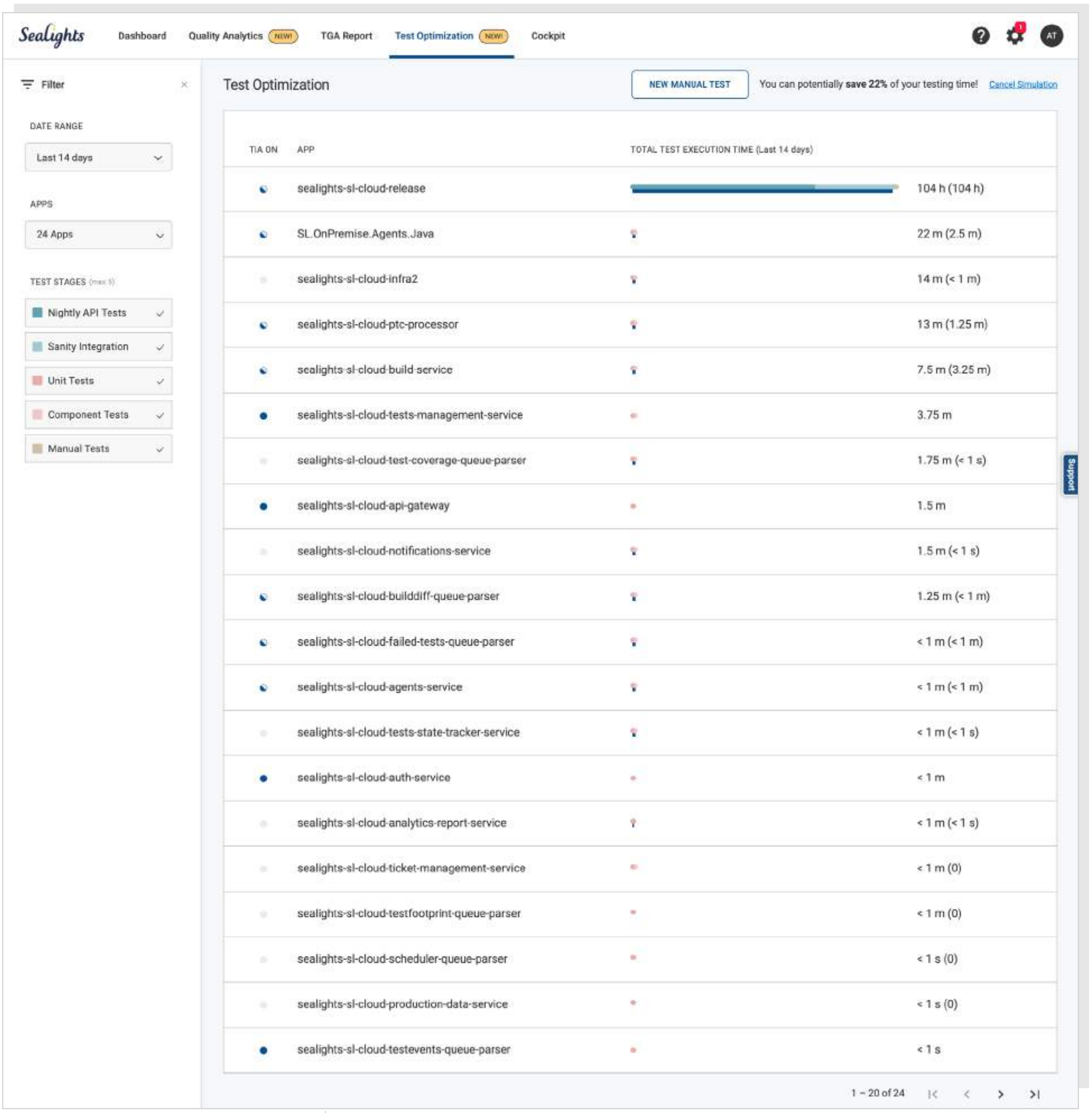
Figure 21: Accessing Test Optimization

UNDERSTANDING THE DATA

The high-level report includes 4 main levels of information:

1. **Summary view** — lists all of your applications, sorted by the total time saved (for each app and each test stage).
2. **Application details** — lists the app name, branch name, analysis period, and the relevant test stage.

3. **Summary** — includes aggregate information:
    a. Average run-time and Average # of tests (with or without tests selection)
    b. Average time saved and Average # of tests reduced (with or without tests selection)
    c. Estimated total time saved for the analysis period



Test Optimization

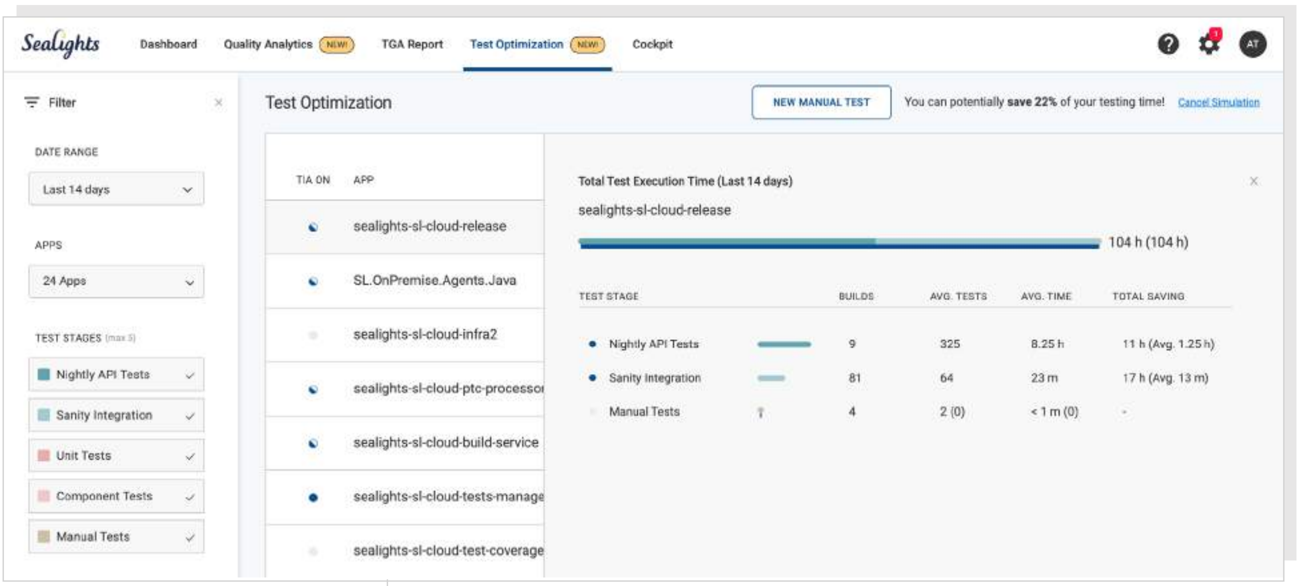
You can potentially **save 22%** of your testing time!

TIA ON	APP	TOTAL TEST EXECUTION TIME (Last 14 days)	
◐	sealights-sl-cloud-release		104 h (104 h)
◐	SL.OnPremise.Agents.Java		22 m (2.5 m)
	sealights-sl-cloud-infra2		14 m (< 1 m)
◐	sealights-sl-cloud-ptc-processor		13 m (1.25 m)
◐	sealights-sl-cloud-build-service		7.5 m (3.25 m)
●	sealights-sl-cloud-tests-management-service		3.75 m
	sealights-sl-cloud-test-coverage-queue-parser		1.75 m (< 1 s)
●	sealights-sl-cloud-api-gateway		1.5 m
	sealights-sl-cloud-notifications-service		1.5 m (< 1 s)
◐	sealights-sl-cloud-builddiff-queue-parser		1.25 m (< 1 m)
◐	sealights-sl-cloud-failed-tests-queue-parser		< 1 m (< 1 m)
◐	sealights-sl-cloud-agents-service		< 1 m (< 1 m)
	sealights-sl-cloud-tests-state-tracker-service		< 1 m (< 1 s)
●	sealights-sl-cloud-auth-service		< 1 m
	sealights-sl-cloud-analytics-report-service		< 1 m (< 1 s)
	sealights-sl-cloud-ticket-management-service		< 1 m (0)
	sealights-sl-cloud-testfootprint-queue-parser		< 1 m (0)
	sealights-sl-cloud-scheduler-queue-parser		< 1 s (0)
	sealights-sl-cloud-production-data-service		< 1 s (0)
●	sealights-sl-cloud-testevents-queue-parser		< 1 s

1 – 20 of 24

Figure 22: Test Optimization (High-level Report)

4. **Test Execution details** - including per-build information:
   a. Build number and build date
   b. Recommended tests that should be run
   c. Estimate of the run time for each test



Test Optimization

You can potentially **save 22%** of your testing time!

TIA ON	APP
◐	sealights-sl-cloud-release
◐	SL.OnPremise.Agents.Java
○	sealights-sl-cloud-infra2
◐	sealights-sl-cloud-ptc-processor
◐	sealights-sl-cloud-build-service
●	sealights-sl-cloud-tests-manage
○	sealights-sl-cloud-test-coverage

Total Test Execution Time (Last 14 days)

sealights-sl-cloud-release

104 h (104 h)

TEST STAGE	BUILDS	AVG. TESTS	AVG. TIME	TOTAL SAVING
● Nightly API Tests	9	325	8.25 h	11 h (Avg. 1.25 h)
● Sanity Integration	81	64	23 m	17 h (Avg. 13 m)
Manual Tests	4	2 (0)	< 1 m (0)	-

Figure 23: Test Optimization (Test Execution Details)

Test Recommendations Report

This report presents a list of the impacted tests for each build.

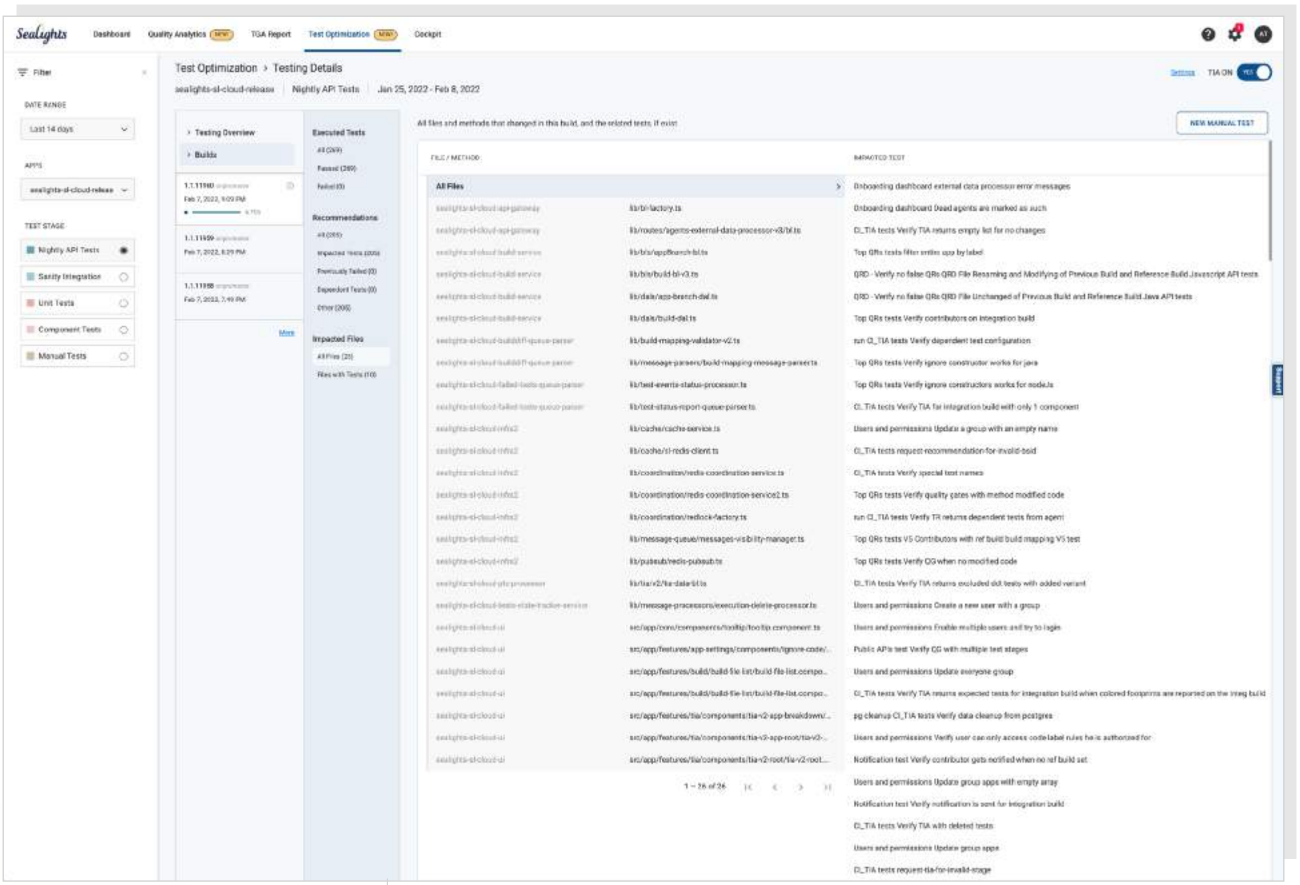
NOTE: This information is also available through the SeaLights API.

USING THE DATA

Identifying Optimization Opportunities

While the TIA page shows the applications in two separate lists, separated by which application has test stages with TIA on, the entire application list is always shown, with filters on the left side of the page.

The applications are marked with a blue circle (TIA fully on), gray circle (TIA off) or partial circle (some of the test stages are with TIA on). Use the status to identify which applications have some optimization applied to them (which you should still review regularly) and which applications are candidates for optimization.


Dashboard · Quality Analytics NEW · TGA Report · Test Optimization NEW · Cockpit

Filter

DATE RANGE: Last 14 days

APPS: sealights-sl-cloud-release

TEST STAGE:
- 🟦 Nightly API Tests ●
- Sanity Integration ○
- Unit Tests ○
- Component Tests ○
- Manual Tests ○

Test Optimization › Testing Details

sealights-sl-cloud-release | Nightly API Tests | Jan 25, 2022 - Feb 8, 2022

Settings · TIA ON: YES

› Testing Overview

› Builds
- 1.1.11960 – Feb 7, 2022, 9:09 PM
- 1.1.11959 – Feb 7, 2022, 8:29 PM
- 1.1.11958 – Feb 7, 2022, 7:49 PM

More

Executed Tests
- All (269)
- Passed (269)
- Failed (0)

Recommendations
- All (285)
- Impacted Tests (220)
- Previously Failed (0)
- Dependent Tests (0)
- Other (205)

Impacted Files
- All Files (25)
- Files with Tests (10)

All files and methods that changed in this build, and the related tests, if exist

NEW MANUAL TEST

FILE / METHOD		IMPACTED TEST
All Files		Onboarding dashboard external data processor error messages
sealights-sl-cloud-api-gateway	lib/bl-factory.ts	Onboarding dashboard Dead agents are marked as such
sealights-sl-cloud-api-gateway	lib/routes/agents-external-data-processor-v3/bl.ts	CI_TIA tests Verify TIA returns empty list for no changes
sealights-sl-cloud-build-service	lib/bls/appBranch-bl.ts	Top QRs tests filter entire app by label
sealights-sl-cloud-build-service	lib/bls/build-bl-v3.ts	QRD - Verify no false QRs QRD File Renaming and Modifying of Previous Build and Reference Build Javascript API tests
sealights-sl-cloud-build-service	lib/dals/app-branch-dal.ts	QRD - Verify no false QRs QRD File Unchanged of Previous Build and Reference Build Java API tests
sealights-sl-cloud-build-service	lib/dals/build-dal.ts	Top QRs tests Verify contributors on integration build
sealights-sl-cloud-builddiff-queue-parser	lib/build-mapping-validator-v2.ts	run CI_TIA tests Verify dependent test configuration
sealights-sl-cloud-builddiff-queue-parser	lib/message-parsers/build-mapping-message-parser.ts	Top QRs tests Verify ignore constructor works for java
sealights-sl-cloud-failed-tests-queue-parser	lib/test-events-status-processor.ts	Top QRs tests Verify ignore constructors works for nodeJs
sealights-sl-cloud-failed-tests-queue-parser	lib/test-status-report-queue-parser.ts	CI_TIA tests Verify TIA for integration build with only 1 component
sealights-sl-cloud-infra2	lib/cache/cache-service.ts	Users and permissions Update a group with an empty name
sealights-sl-cloud-infra2	lib/cache/sl-redis-client.ts	CI_TIA tests request-recommendation-for-invalid-bsid
sealights-sl-cloud-infra2	lib/coordination/redis-coordination-service.ts	CI_TIA tests Verify special test names
sealights-sl-cloud-infra2	lib/coordination/redis-coordination-service2.ts	Top QRs tests Verify quality gates with method modified code
sealights-sl-cloud-infra2	lib/coordination/redlock-factory.ts	run CI_TIA tests Verify TR returns dependent tests from agent
sealights-sl-cloud-infra2	lib/message-queue/messages-visibility-manager.ts	Top QRs tests V5 Contributors with ref build build mapping V5 test
sealights-sl-cloud-infra2	lib/pubsub/redis-pubsub.ts	Top QRs tests Verify QG when no modified code
sealights-sl-cloud-gtc-processor	lib/tia/v2/tia-data-bl.ts	CI_TIA tests Verify TIA returns excluded skit tests with added variant
sealights-sl-cloud-tests-state-tracker-service	lib/message-processors/execution-delete-processor.ts	Users and permissions Create a new user with a group
sealights-sl-cloud-ui	src/app/core/components/tooltip/tooltip.component.ts	Users and permissions Enable multiple users and try to login
sealights-sl-cloud-ui	src/app/features/app-settings/components/ignore-code/...	Public APIs test Verify QG with multiple test stages
sealights-sl-cloud-ui	src/app/features/build/build-file-list/build-file-list.compo...	Users and permissions Update everyone group
sealights-sl-cloud-ui	src/app/features/build/build-file-list/build-file-list.compo...	CI_TIA tests Verify TIA returns expected tests for integration build when colored footprints are reported on the integ build
sealights-sl-cloud-ui	src/app/features/tia/components/tia-v2-app-breakdown/...	pg cleanup CI_TIA tests Verify data cleanup from postgres
sealights-sl-cloud-ui	src/app/features/tia/components/tia-v2-app-root/tia-v2-...	Users and permissions Verify user can only access code label rules he is authorized for
sealights-sl-cloud-ui	src/app/features/tia/components/tia-v2-root/tia-v2-root...	Notification test Verify contributor gets notified when no ref build set
		Users and permissions Update group apps with empty array
		Notification test Verify notification is sent for integration build
		CI_TIA tests Verify TIA with deleted tests
		Users and permissions Update group apps
		CI_TIA tests request-tia-for-invalid-stage

1 – 26 of 26

Figure 24: Test Optimization (Test Recommendations)

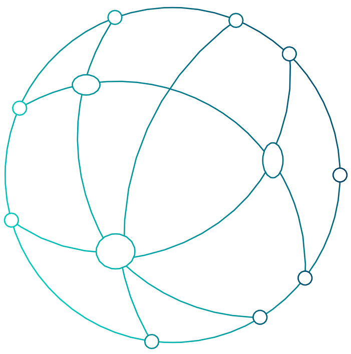
Execute Only Recommended Tests

You should direct SeaLights to execute only the tests that are recommended, excluding any tests that SeaLights has excluded because the tests do not cover code that is new or has been modified, tests that have passed in the previous run, etc..

Regularly Execute “Full Run” Tests

However, you should find time to regularly run all tests to ensure that no tests are ever overlooked.

SUMMARY



In this guide we have shared some recommendations for best practices for using SeaLights and the steps to provide a foundation for supporting and accelerating our customer's quality strategies, specifically:

- Identify New/Modified Code in the background without impacting existing development
- Guide the software development team to the best decisions
- Determine release readiness, identify and prevent untested code changes making it to production
- Direct teams to where to develop and execute the minimal number of tests

These best practices are quite generic, and we recognize that customers have specific requirements that will need these best practices to be adapted. We welcome feedback from our customers on their experiences and requirements so we can improve SeaLights's support for additional common use cases out of the box, and evolve this document to include specific customer experiences to share with other organizations with similar challenges.

USEFUL LINKS

SeaLights Home Page:
https://www.sealights.io/

White Papers and e-Books:
https://www.sealights.io/learn/

Webinars:
https://www.sealights.io/webinars/

SeaLights Blog:
https://www.sealights.io/blog/

What's New at SeaLights:
https://www.sealights.io/whats-new/

How-to Articles:
https://sealights.atlassian.net/wiki/spaces/SUP/pages/1376261/How-to+articles

Learn About SeaLights:
https://sealights.atlassian.net/wiki/spaces/SUP/pages/802652190/Learn+about+SeaLights

Sealights