Diff coverage

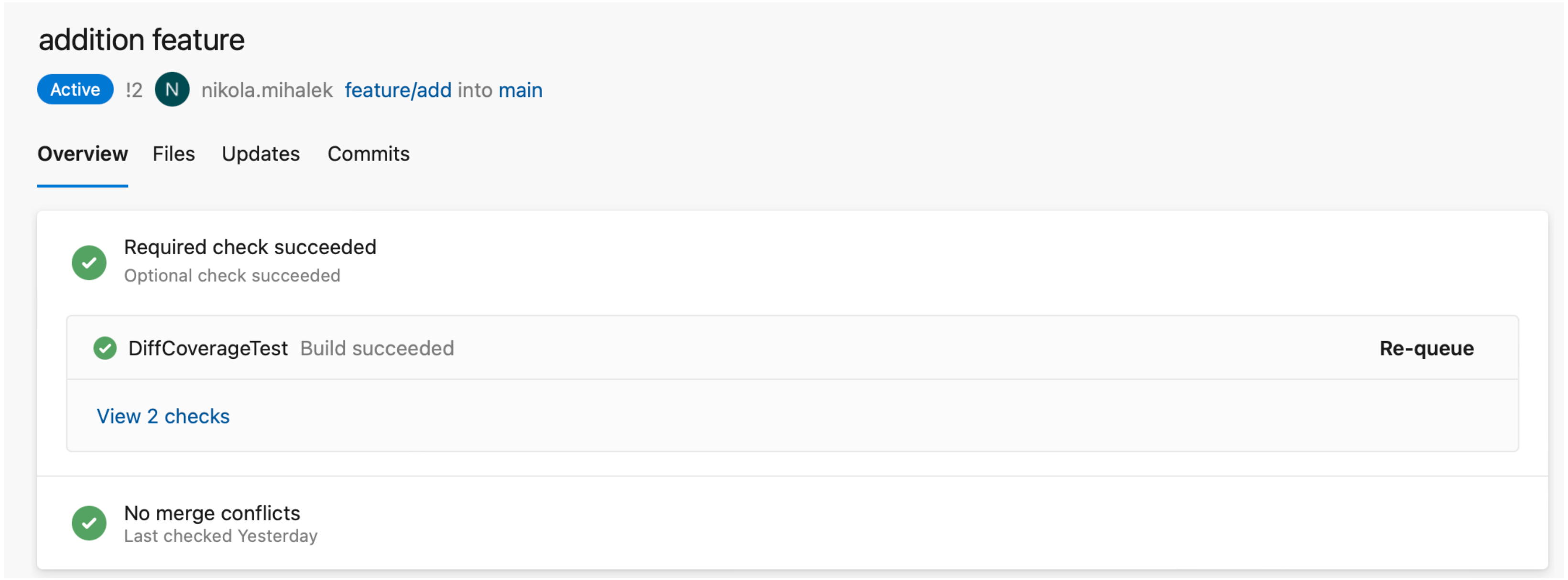
Nikola Mihalek

An overview of topics we will cover today

Looks like a lot, but we’ll be quick!

- Diff coverage or PR coverage
- It means code test coverage for a given git diff, or in other words:
- Code test coverage for changes!

Diff..huh?



addition feature

Active !2 N nikola.mihalek feature/add into main

Overview Files Updates Commits

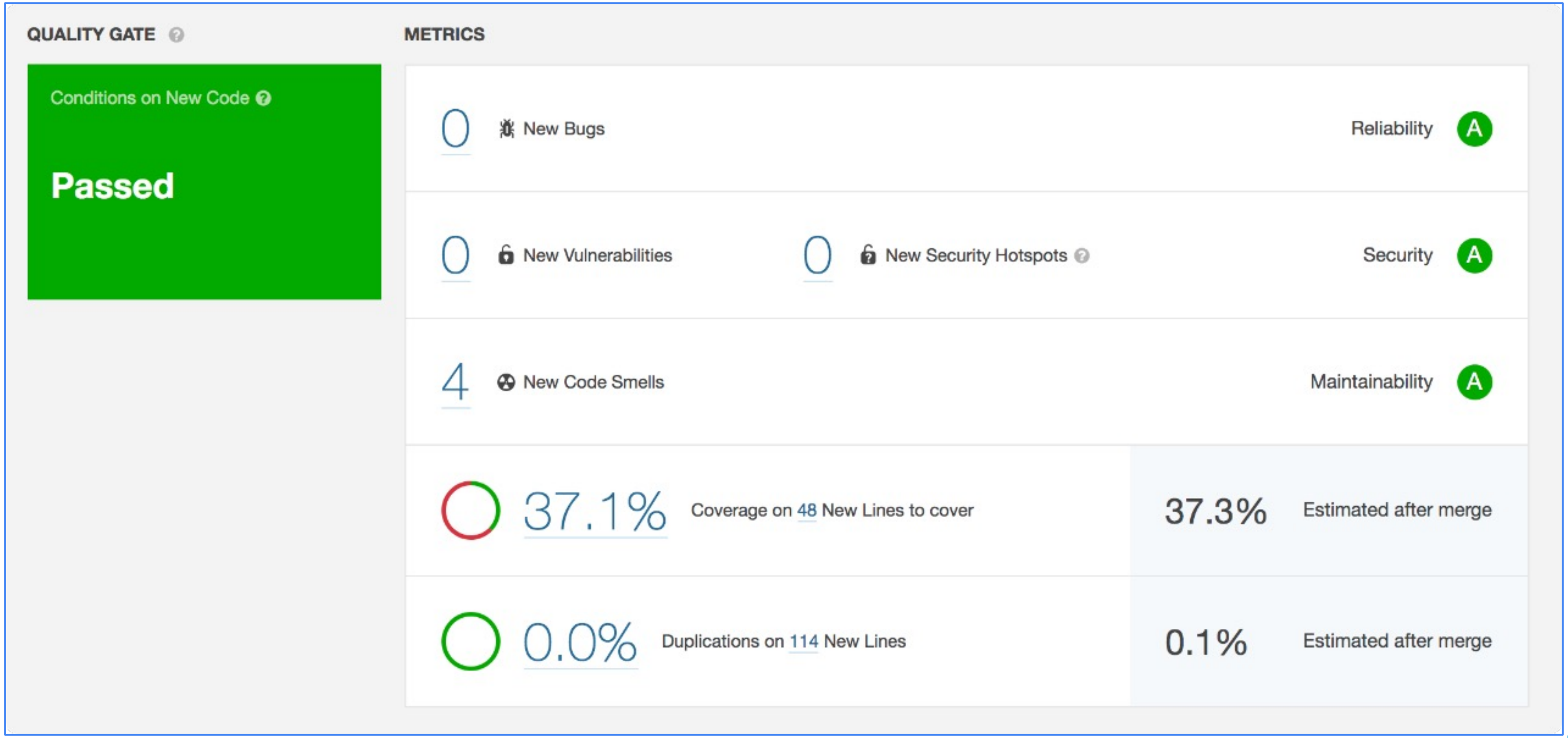
Required check succeeded
Optional check succeeded

DiffCoverageTest Build succeeded Re-queue

View 2 checks

No merge conflicts
Last checked Yesterday

Diff..huh?



QUALITY GATE

Conditions on New Code

Passed

METRICS

0	New Bugs			Reliability	A
0	New Vulnerabilities	0	New Security Hotspots	Security	A
4	New Code Smells			Maintainability	A
37.1%	Coverage on 48 New Lines to cover			37.3%	Estimated after merge
0.0%	Duplications on 114 New Lines			0.1%	Estimated after merge

- Quality gates, mostly!
- But also as a reminder on what's left to cover with tests
- It's a great tool to increase test coverage on a project
- Used as part of a CI pipeline

What’s it used for anyway? – When’s it triggered

- Diff code coverage checks are usually triggered as part of a CI pipeline
- It’s a step after the coverage report has been generated
- It uses the coverage report and the git diff to calculate the total coverage for a given diff (PR)

What’s it used for anyway? – What are quality gates

- We all strive for great testable code, but thinking about it all the time is difficult
- Quality gates are a set of requirements needed for a PR to be merged
- These can be anything:
  - Architecture
  - Code style
  - Test coverage
  - Vulnerabilities / code smells
  - Number of reviewers / specific reviewers
- They should be defined by the team
- We will be focusing on test coverage today
- A good middle ground for a test coverage gate is ~80%

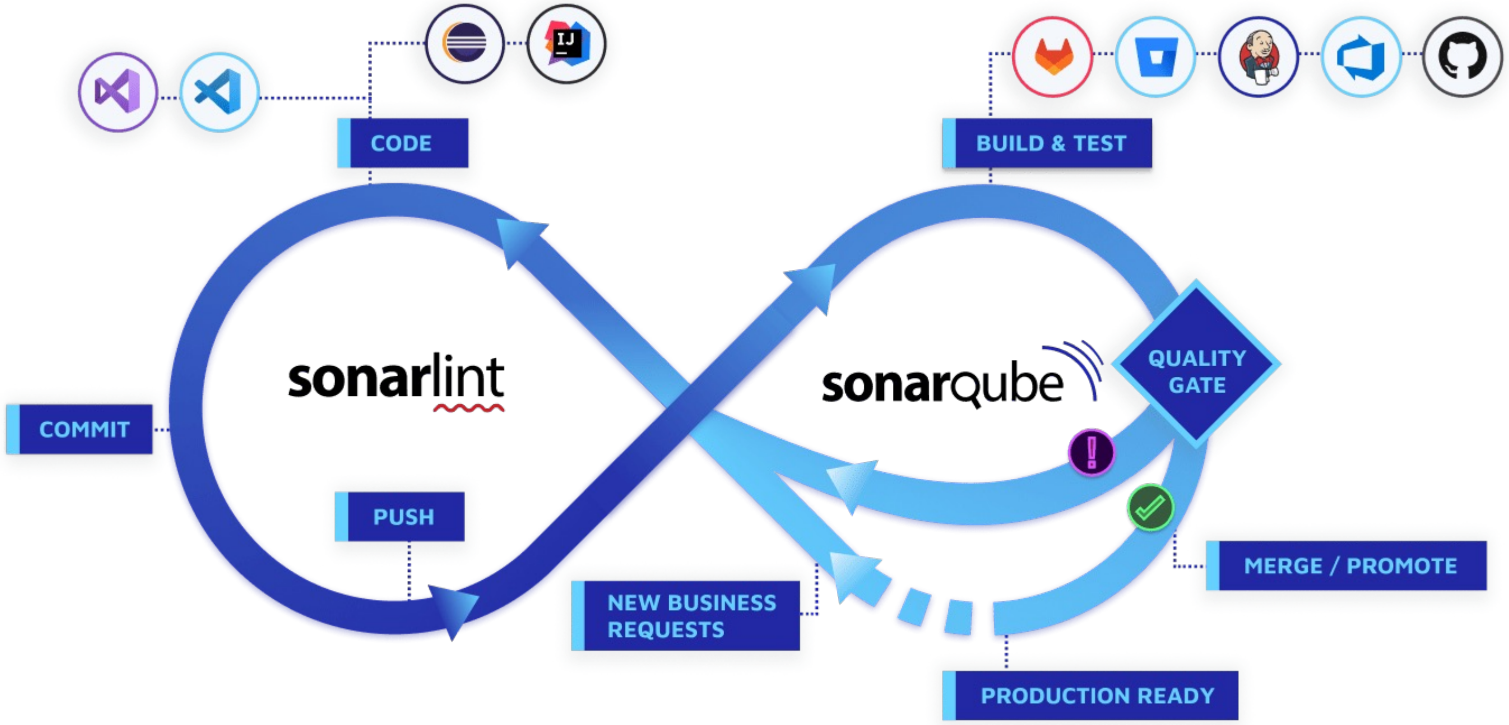
Let’s look at a simple example app!

Calculator app showcase

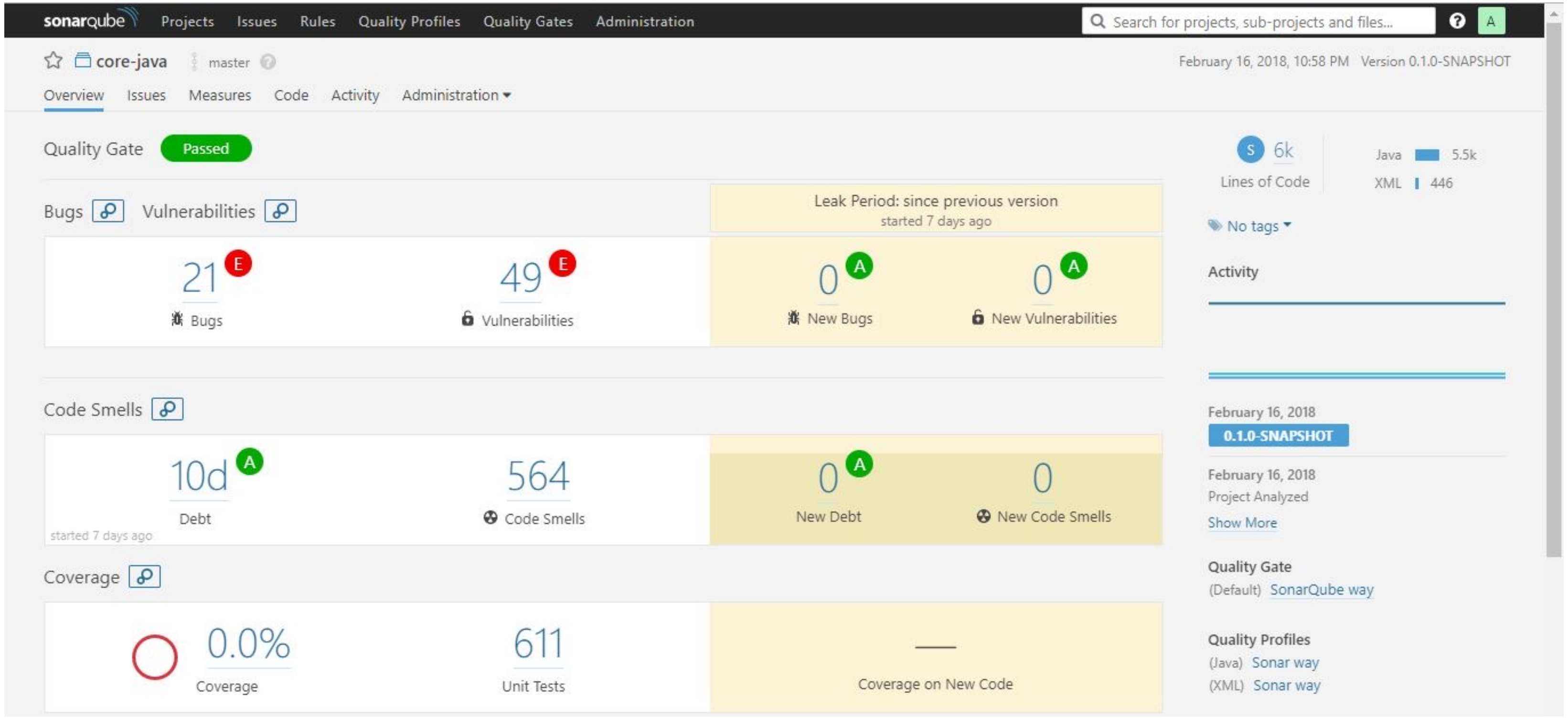
So, what should we use to ensure test coverage for this example?

- The most popular tool on the market
- And for good reason!
- Complete solution:
  - Code coverage
  - Code smells
  - Vulnerabilities
  - Static code analysis
  - Automatic integration with you git server
  - Many, many supported languages
  - Other things I've missed
- So, get this, right? Maybe

Sonar


CODE
BUILD & TEST
sonarlint
sonarqube
QUALITY GATE
COMMIT
PUSH
MERGE / PROMOTE
NEW BUSINESS REQUESTS
PRODUCTION READY

Sonar


sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration

Search for projects, sub-projects and files...

core-java master

February 16, 2018, 10:58 PM Version 0.1.0-SNAPSHOT

Overview Issues Measures Code Activity Administration

Quality Gate Passed

Bugs Vulnerabilities

Leak Period: since previous version
started 7 days ago

21 E Bugs

49 E Vulnerabilities

0 A New Bugs

0 A New Vulnerabilities

Code Smells

10d A Debt

started 7 days ago

564 Code Smells

0 A New Debt

0 New Code Smells

Coverage

0.0% Coverage

611 Unit Tests

— Coverage on New Code

S 6k Lines of Code

Java	🟦	5.5k
XML	▮	446

No tags

Activity

February 16, 2018
0.1.0-SNAPSHOT

February 16, 2018
Project Analyzed

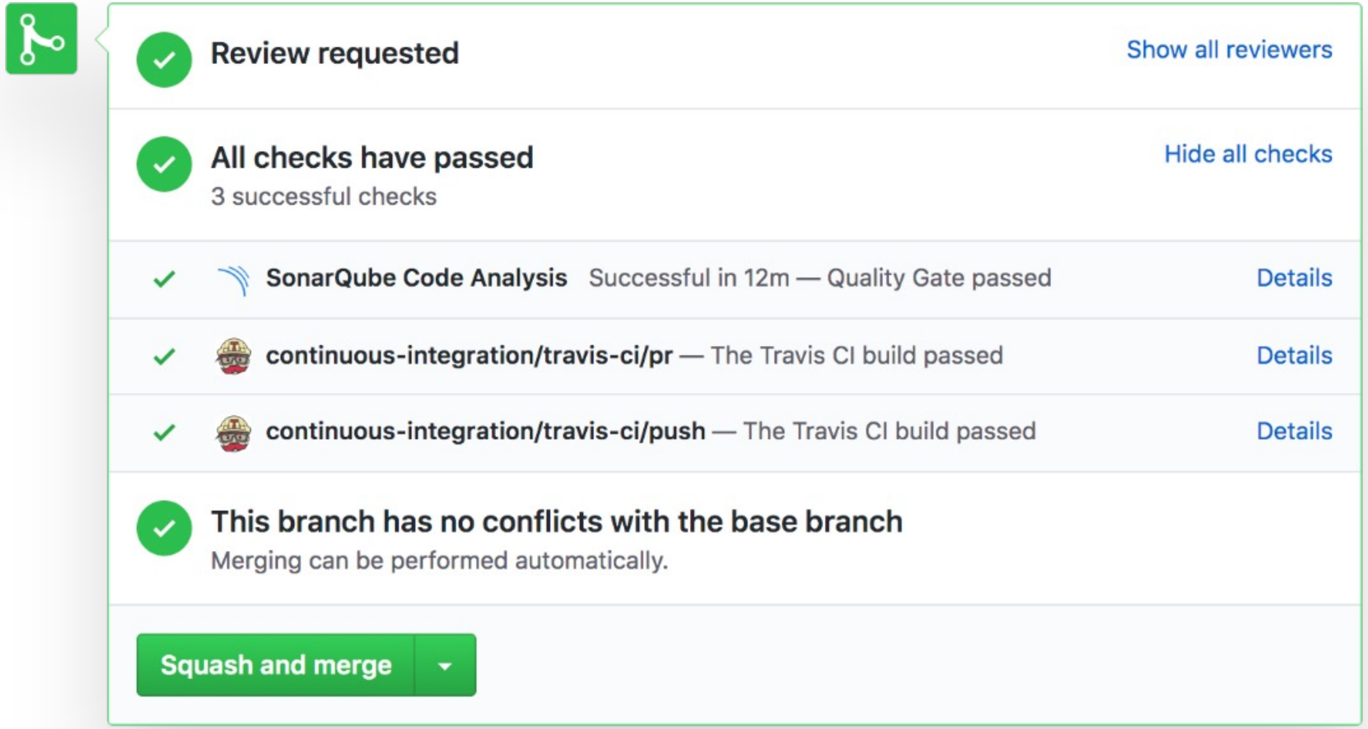
Show More

Quality Gate
(Default) SonarQube way

Quality Profiles
(Java) Sonar way
(XML) Sonar way

HTEC GROUP

Sonar



Review requested

Show all reviewers

All checks have passed

3 successful checks

Hide all checks

✓ SonarQube Code Analysis Successful in 12m — Quality Gate passed Details

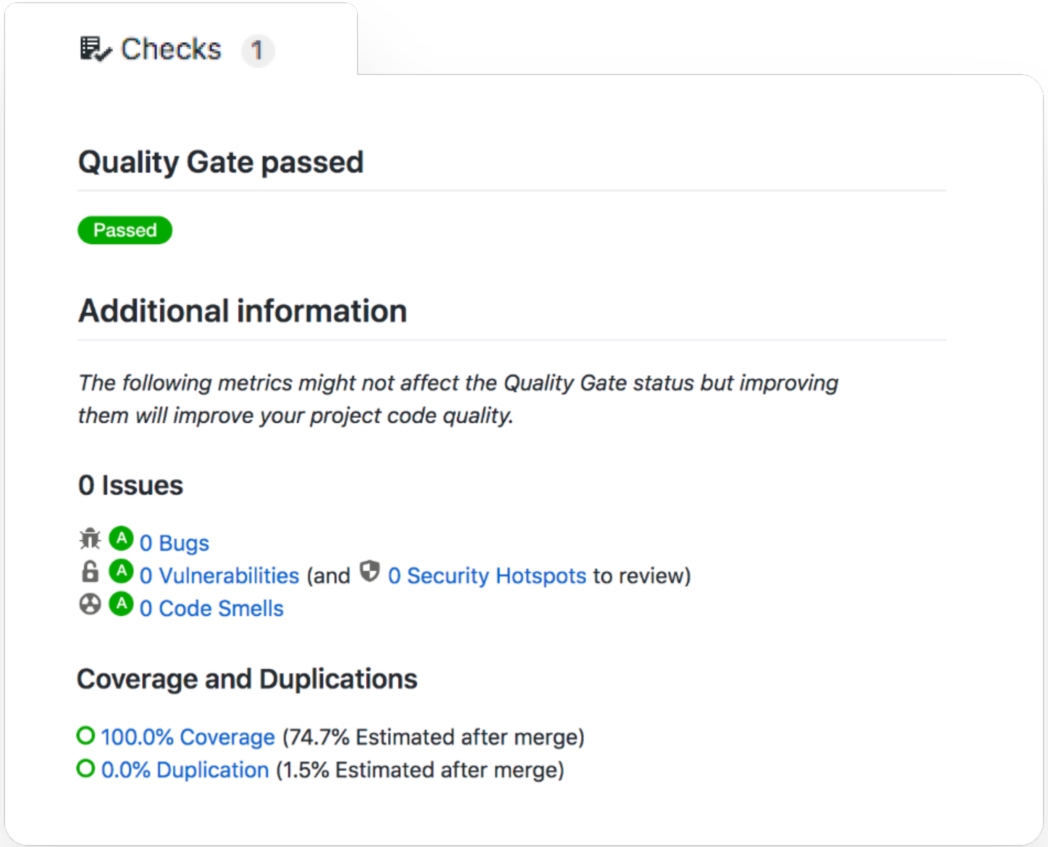
✓ continuous-integration/travis-ci/pr — The Travis CI build passed Details

✓ continuous-integration/travis-ci/push — The Travis CI build passed Details

This branch has no conflicts with the base branch

Merging can be performed automatically.

Sonar



Quality Gate passed

Passed

Additional information

The following metrics might not affect the Quality Gate status but improving them will improve your project code quality.

0 Issues

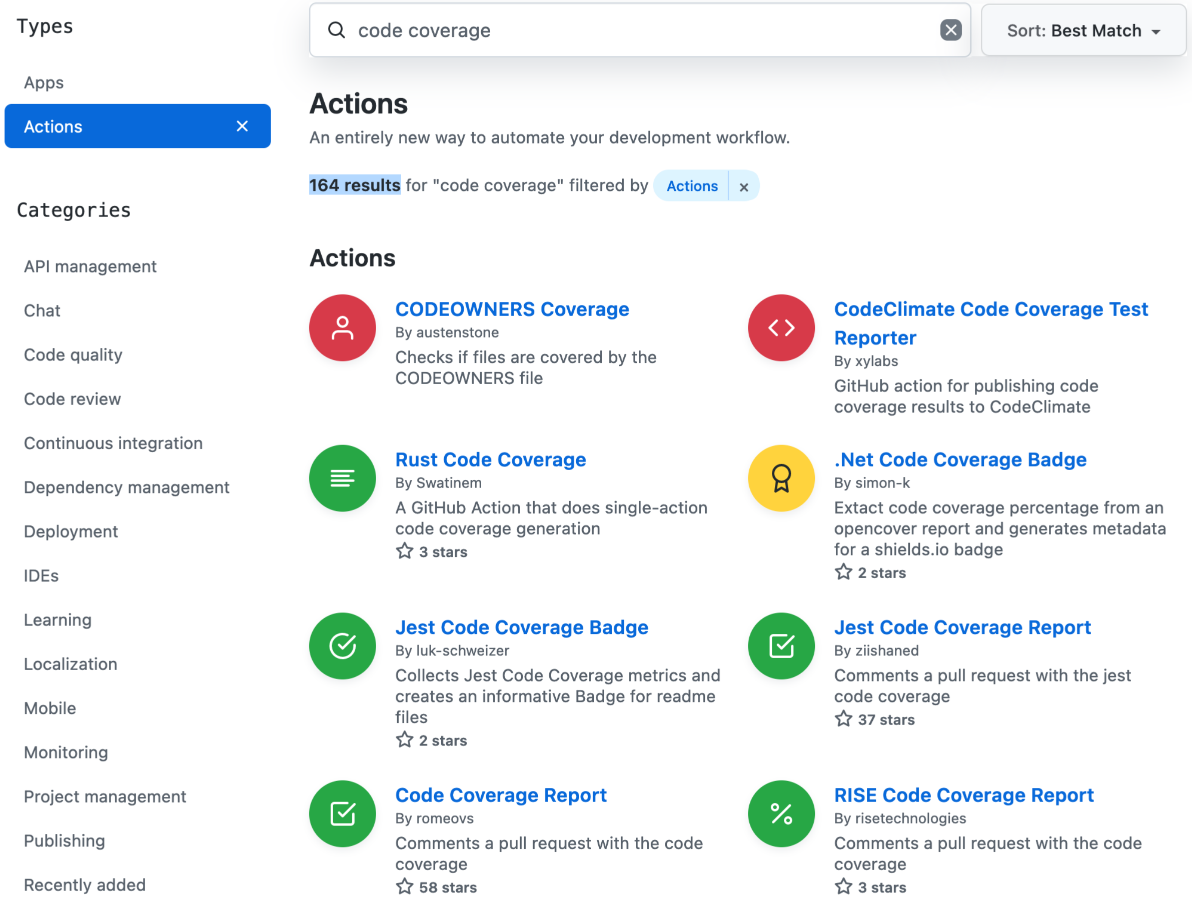
A 0 Bugs
A 0 Vulnerabilities (and 0 Security Hotspots to review)
A 0 Code Smells

Coverage and Duplications

100.0% Coverage (74.7% Estimated after merge)
0.0% Duplication (1.5% Estimated after merge)

- The new kid on the block
- Integrated solution all in one place (if you're using GitHub)
- Not much fiddling around to get it setup
- A good selection of choices, made by the community
- Used as part of a GitHub workflow (CI/CD)

GitHub Actions



Types

Apps

Actions ×

Categories

API management

Chat

Code quality

Code review

Continuous integration

Dependency management

Deployment

IDEs

Learning

Localization

Mobile

Monitoring

Project management

Publishing

Recently added

code coverage

Sort: Best Match

Actions

An entirely new way to automate your development workflow.

164 results for "code coverage" filtered by Actions ×

Actions

CODEOWNERS Coverage
By austenstone
Checks if files are covered by the CODEOWNERS file

CodeClimate Code Coverage Test Reporter
By xylabs
GitHub action for publishing code coverage results to CodeClimate

Rust Code Coverage
By Swatinem
A GitHub Action that does single-action code coverage generation
☆ 3 stars

.Net Code Coverage Badge
By simon-k
Exact code coverage percentage from an opencover report and generates metadata for a shields.io badge
☆ 2 stars

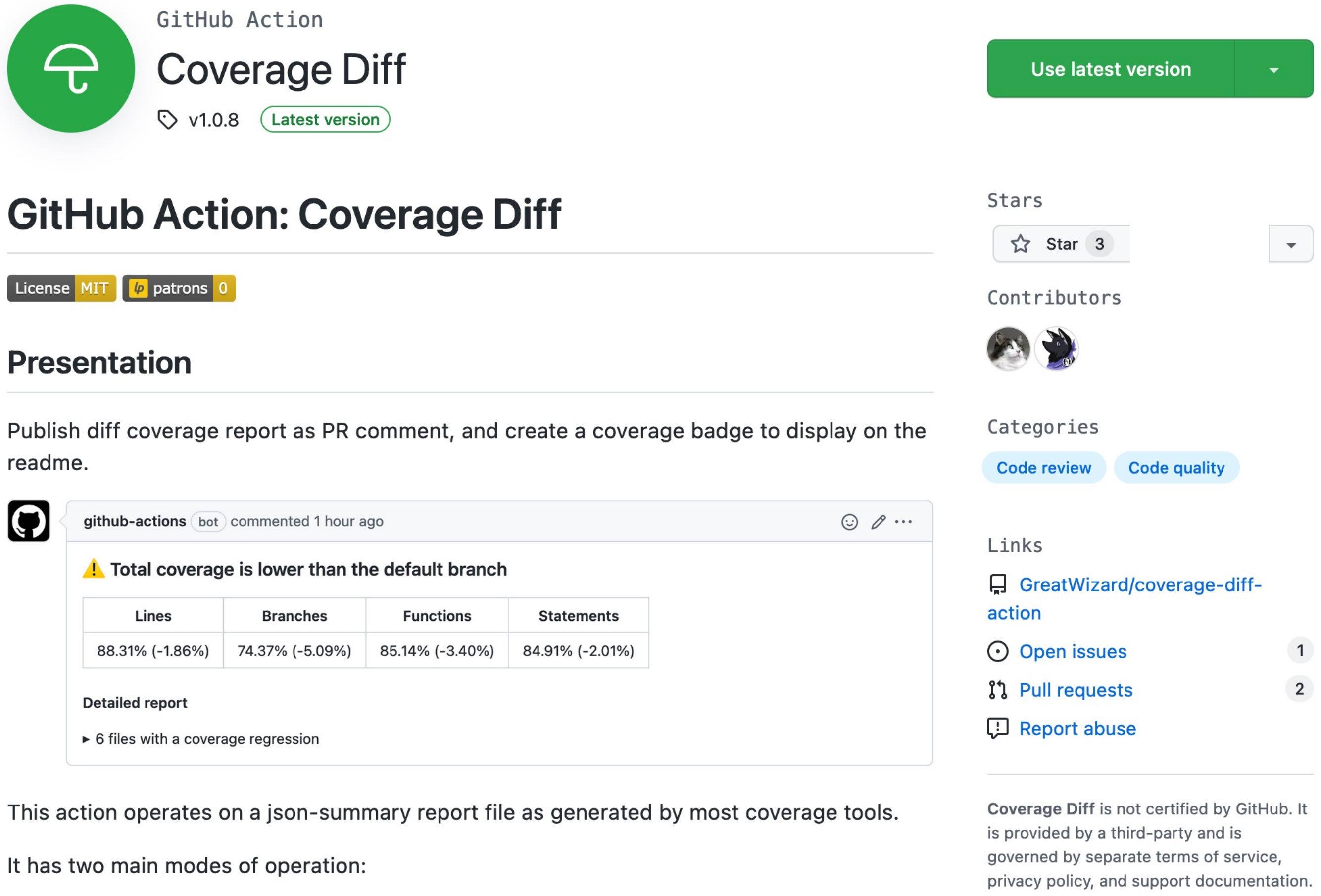
Jest Code Coverage Badge
By luk-schweizer
Collects Jest Code Coverage metrics and creates an informative Badge for readme files
☆ 2 stars

Jest Code Coverage Report
By ziishaned
Comments a pull request with the jest code coverage
☆ 37 stars

Code Coverage Report
By romeovs
Comments a pull request with the code coverage
☆ 58 stars

RISE Code Coverage Report
By risetechnologies
Comments a pull request with the code coverage
☆ 3 stars

GitHub Actions



GitHub Action

Coverage Diff

v1.0.8 Latest version

Use latest version

GitHub Action: Coverage Diff

License MIT | patrons 0

Presentation

Publish diff coverage report as PR comment, and create a coverage badge to display on the readme.

github-actions bot commented 1 hour ago

⚠️ **Total coverage is lower than the default branch**

Lines	Branches	Functions	Statements
88.31% (-1.86%)	74.37% (-5.09%)	85.14% (-3.40%)	84.91% (-2.01%)

Detailed report

▸ 6 files with a coverage regression

This action operates on a json-summary report file as generated by most coverage tools.

It has two main modes of operation:

Stars

Star 3

Contributors

Categories

Code review Code quality

Links

GreatWizard/coverage-diff-action

Open issues 1

Pull requests 2

Report abuse

Coverage Diff is not certified by GitHub. It is provided by a third-party and is governed by separate terms of service, privacy policy, and support documentation.

- So, should you use one of these?
- Absolutely, if you can
- But these two solutions don’t cover every case
- Let’s take a look…

What’s wrong with these

Sonar

- Proprietary, closed-source solution
- Expensive
- There’s no option to have diff coverage analysis integration without paying (or without hosting your own server and violation the license agreement)
- What if the client doesn’t want to pay?
- What if your project is an open-source or hobby project?

GitHub Actions

- Not mature enough
- Many options, but it’s not clear which is optimal
- Limited to using GitHub as your repository
- Limited to using GitHub as your CI
- Limited language support (depending on the option)

What’s wrong with these

Sonar and GitHub Actions are great, but if you can’t use them, for you diff code coverage is:

What’s wrong with these

- DIY – do it yourself
- Can be used with any git provider
- Free
- Not too complicated to setup, if you know how
- I'll show a working example on Azure DevOps

An alternative approach

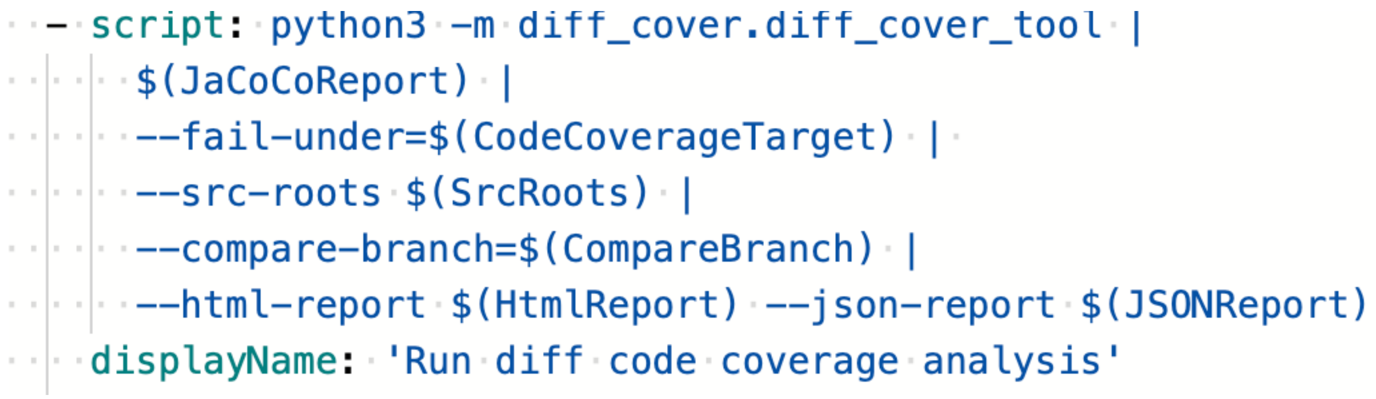
Basic high-level steps:

- Create a coverage report for the whole project
- Generate the diff coverage report using a tool like diff_cover (or any other you find/like)
- Update the PR status (using a HTTP POST request, for example)

ADO diff cover showcase

- Download python module
- Run it and pass in the following params:
    - Test coverage report
    - Source code location(s)
    - Code coverage target (>80% FTW)
    - Compare branch (main or dev)
    - Output report formats and paths
- Two reports are generated: JSON and HTML
- JSON report is used to extract info to update the PR status
- HTML report is in a more human-readable format, so it's used to look pretty

Generating the diff – step from the CI pipeline

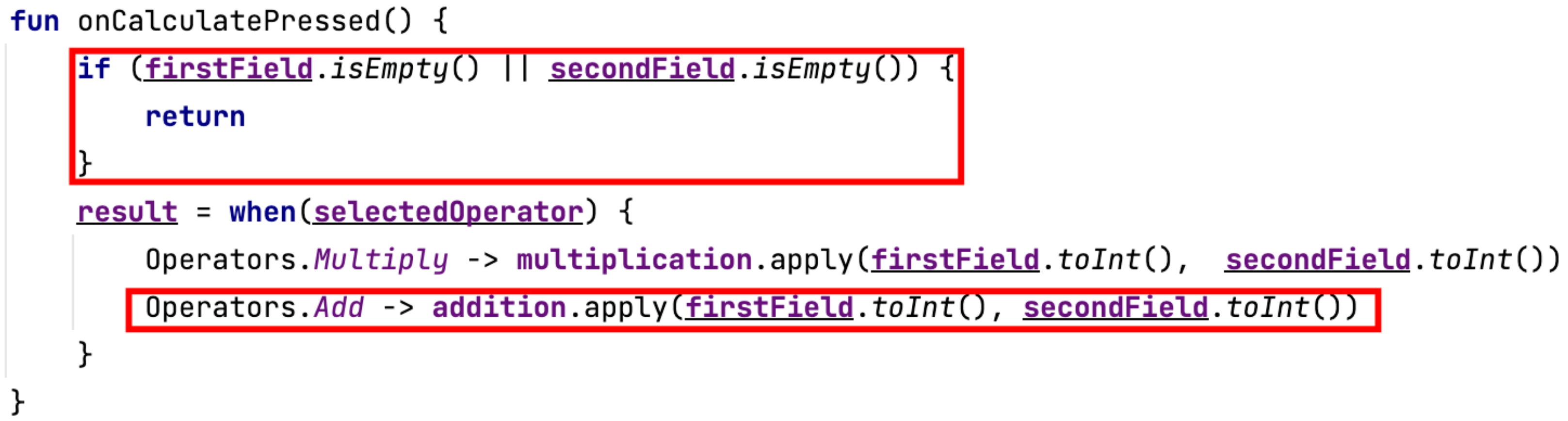


```yaml
  - script: python3 -m diff_cover.diff_cover_tool |
      $(JaCoCoReport) |
      --fail-under=$(CodeCoverageTarget) |
      --src-roots $(SrcRoots) |
      --compare-branch=$(CompareBranch) |
      --html-report $(HtmlReport) --json-report $(JSONReport)
    displayName: 'Run diff code coverage analysis'
```

Generating the diff – actual changes part 1

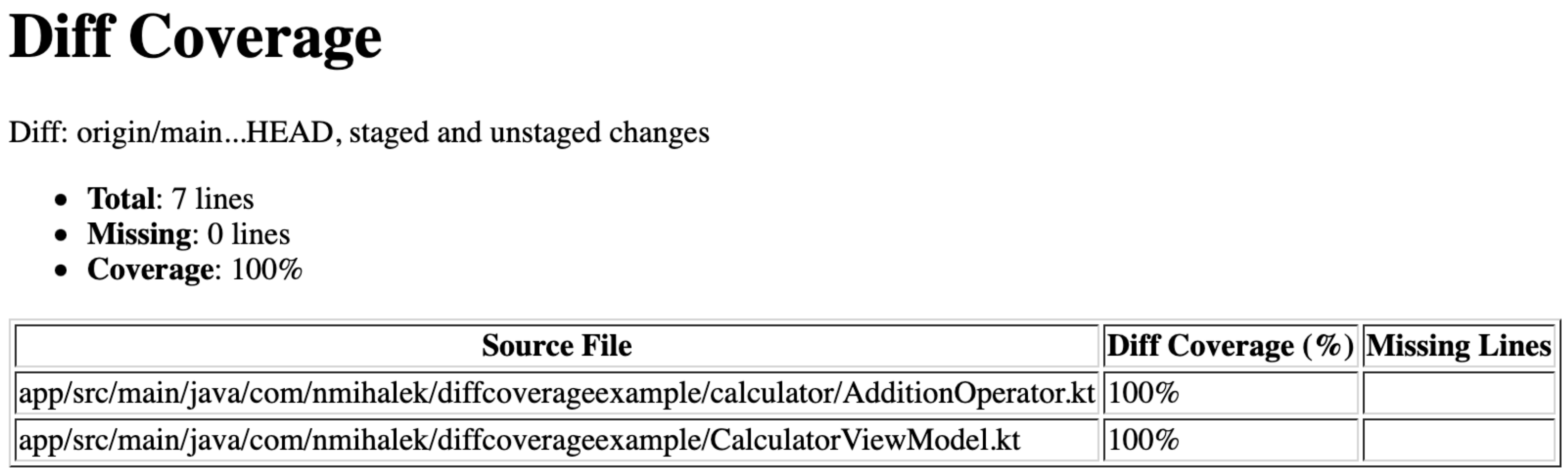
```
class AdditionOperator @Inject constructor(): Operator {
    override fun apply(first: Int, second: Int): Int = first +  second
}
```

Generating the diff – actual changes part 2



```
fun onCalculatePressed() {
    if (firstField.isEmpty() || secondField.isEmpty()) {
        return
    }
    result = when(selectedOperator) {
        Operators.Multiply -> multiplication.apply(firstField.toInt(),  secondField.toInt())
        Operators.Add -> addition.apply(firstField.toInt(), secondField.toInt())
    }
}
```

Generating the diff – HTML report



Diff Coverage

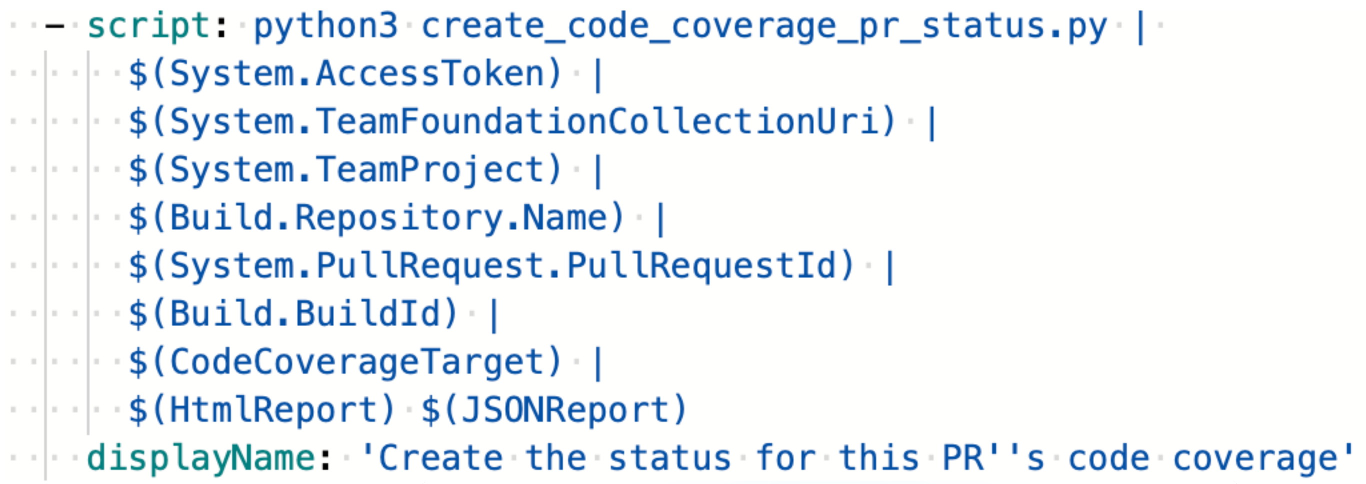
Diff: origin/main...HEAD, staged and unstaged changes

- **Total**: 7 lines
- **Missing**: 0 lines
- **Coverage**: 100%

Source File	Diff Coverage (%)	Missing Lines
app/src/main/java/com/nmihalek/diffcoverageexample/calculator/AdditionOperator.kt	100%	
app/src/main/java/com/nmihalek/diffcoverageexample/CalculatorViewModel.kt	100%	

- We extract the info we need from the generated JSON report
- Now that we have the diff, we need to update our PR with the diff report information
    - Pass/fail status
    - Coverage percentage
- I’ll use a Python client, but could also use a HTTP POST

Updating the PR status – step from the CI pipeline

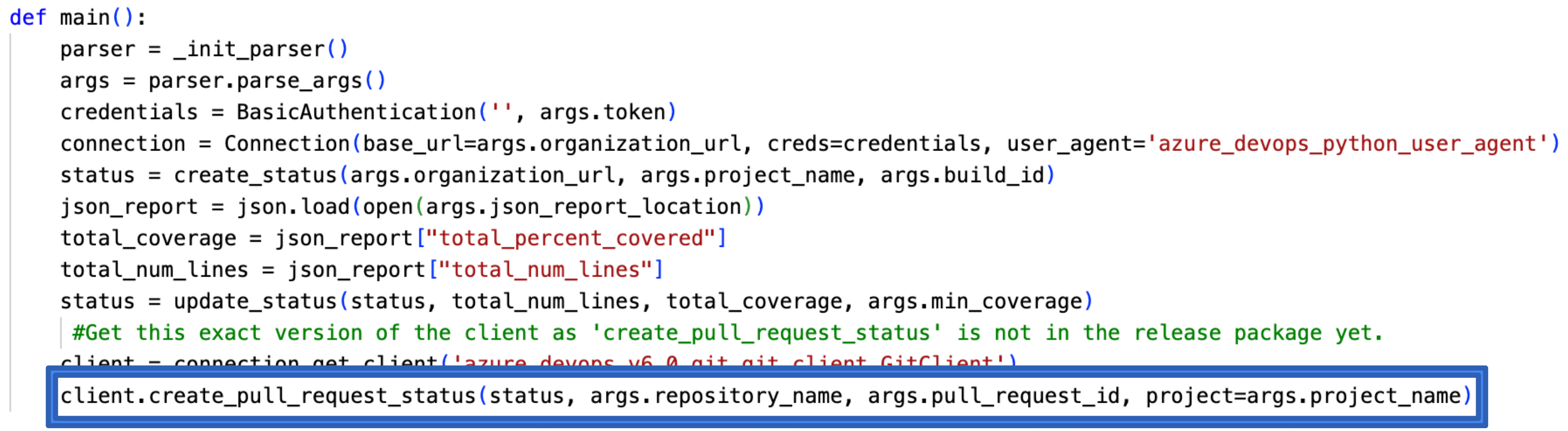


```yaml
  - script: python3 create_code_coverage_pr_status.py |
        $(System.AccessToken) |
        $(System.TeamFoundationCollectionUri) |
        $(System.TeamProject) |
        $(Build.Repository.Name) |
        $(System.PullRequest.PullRequestId) |
        $(Build.BuildId) |
        $(CodeCoverageTarget) |
        $(HtmlReport) $(JSONReport)
    displayName: 'Create the status for this PR''s code coverage'
```

Updating the PR status – Python code

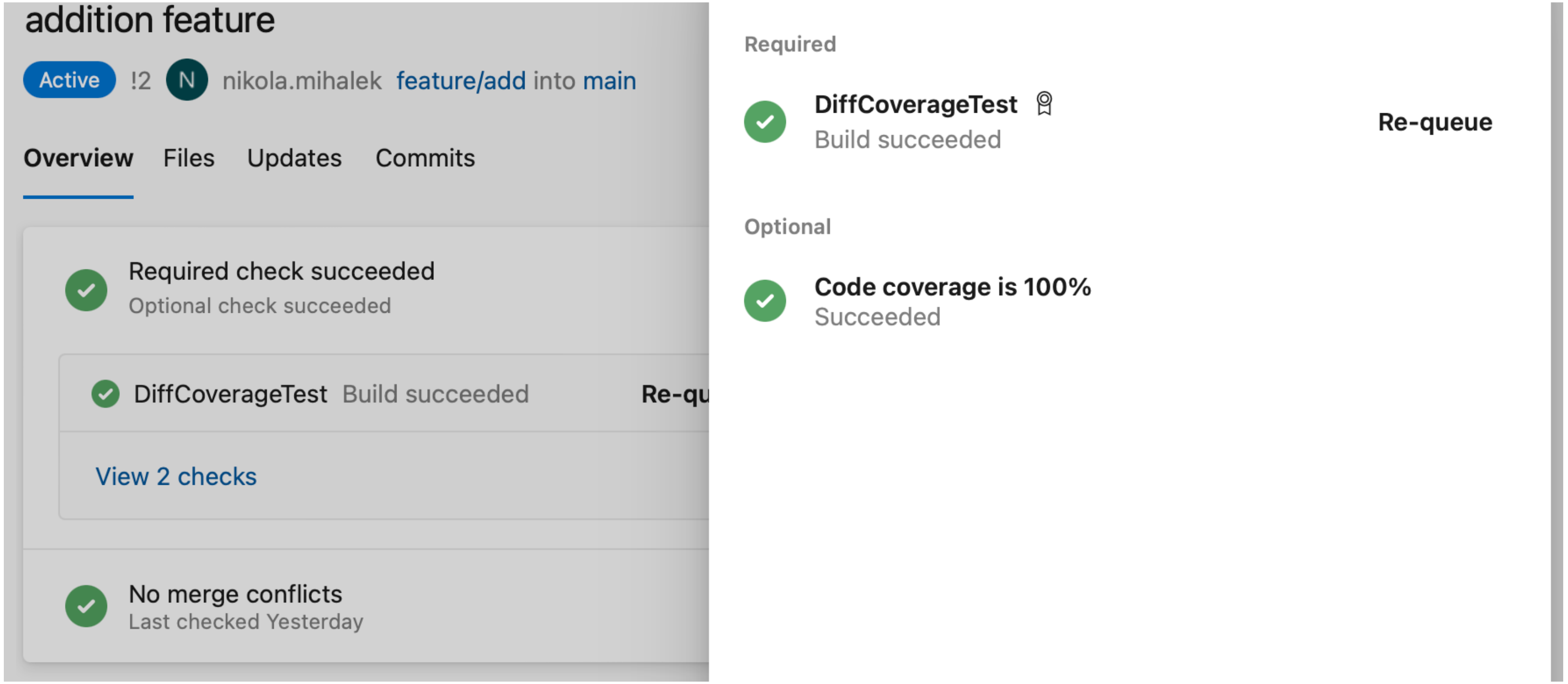
```python
def main():
    parser = _init_parser()
    args = parser.parse_args()
    credentials = BasicAuthentication('', args.token)
    connection = Connection(base_url=args.organization_url, creds=credentials, user_agent='azure_devops_python_user_agent')
    status = create_status(args.organization_url, args.project_name, args.build_id)
    json_report = json.load(open(args.json_report_location))
    total_coverage = json_report["total_percent_covered"]
    total_num_lines = json_report["total_num_lines"]
    status = update_status(status, total_num_lines, total_coverage, args.min_coverage)
     #Get this exact version of the client as 'create_pull_request_status' is not in the release package yet.
    client = connection.get_client('azure.devops.v6_0.git.git_client.GitClient')
    client.create_pull_request_status(status, args.repository_name, args.pull_request_id, project=args.project_name)
```

Updating the PR status – Python code



```python
def main():
    parser = _init_parser()
    args = parser.parse_args()
    credentials = BasicAuthentication('', args.token)
    connection = Connection(base_url=args.organization_url, creds=credentials, user_agent='azure_devops_python_user_agent')
    status = create_status(args.organization_url, args.project_name, args.build_id)
    json_report = json.load(open(args.json_report_location))
    total_coverage = json_report["total_percent_covered"]
    total_num_lines = json_report["total_num_lines"]
    status = update_status(status, total_num_lines, total_coverage, args.min_coverage)
     #Get this exact version of the client as 'create_pull_request_status' is not in the release package yet.
    client = connection.get_client('azure.devops.v6_0.git.git_client.GitClient')
    client.create_pull_request_status(status, args.repository_name, args.pull_request_id, project=args.project_name)
```

Updating the PR status - result



addition feature

Active !2 N nikola.mihalek feature/add into main

Overview Files Updates Commits

✓ Required check succeeded
Optional check succeeded

✓ DiffCoverageTest Build succeeded Re-qu

View 2 checks

✓ No merge conflicts
Last checked Yesterday

Required

✓ DiffCoverageTest
Build succeeded
Re-queue

Optional

✓ Code coverage is 100%
Succeeded

- Code test coverage is an important metric for ensuring code quality
- Diff coverage is a great way to ensure quality
- A good middle ground is ~80% new/modified code covered by tests
- Use Sonar if you can (it really rocks)
- GitHub Actions is nice, but it's early days + you'll need to host your code on GitHub
- If you're using BitBucket, GitLab, Azure DevOps, or any other git repo and you're not paying for Sonar, this alternative approach is for you

References

- Sonar
- GitHub Actions
- Diff-cover
- Azure DevOps pipelines
- Popularity of programming languages
- https://github.com/nmihalek/diff-cover-example

Q&A

Thank you for your attention!