Technical Debt

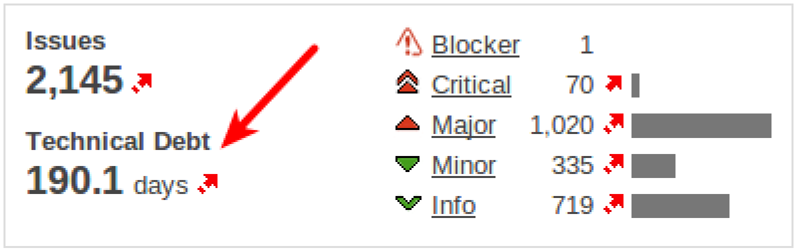
The computation of technical debt is based on the SQALE (Software Quality Assessment based on Lifecycle Expectations) methodology.

SQALE is a methodology that was developed by inspearit and then open sourced. If you read the documentation at sqale.org, you'll see that it's about "organising the non-functional requirements that relate to the code's quality." In the SonarQube implementation of the SQALE method, those non-functional requirements are the coding rules in your quality profile

Yes, the SonarQube implementation of SQALE is based solely on rules and issues. That means that if you want to manage all your technical debt with SQALE, you'll first need to enable the rules in the Common SonarQube repository that flag:

- Duplicated blocks
- Failed unit tests
- Insufficient branch coverage by unit tests
- Insufficient comment density
- Insufficient line coverage by unit tests
- Skipped unit tests

Those rules are in the Common SonarQube repository because they're common to all languages. Once you've got them enabled, you can track every quality flaw as an issue, and you're ready to track technical debt, which the SQALE method measures in days.

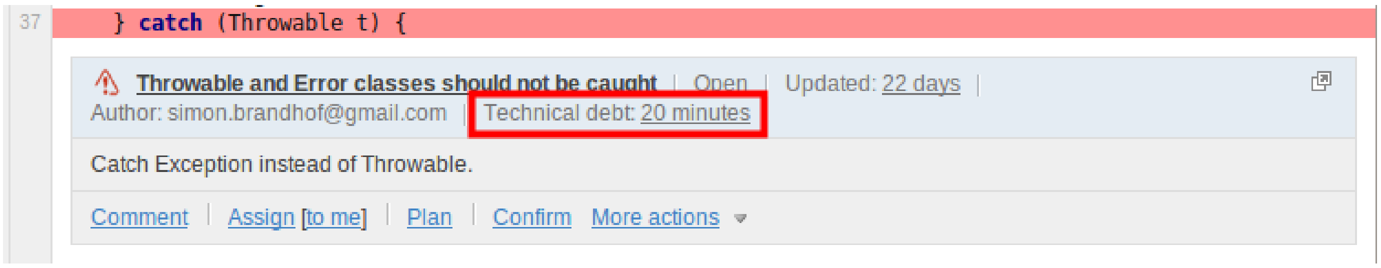


Issues
2,145

Technical Debt
190.1 days

Blocker	1
Critical	70
Major	1,020
Minor	335
Info	719

Those day measurements are made by summing the technical debt accrued for each issue, which you can see in the issue block:



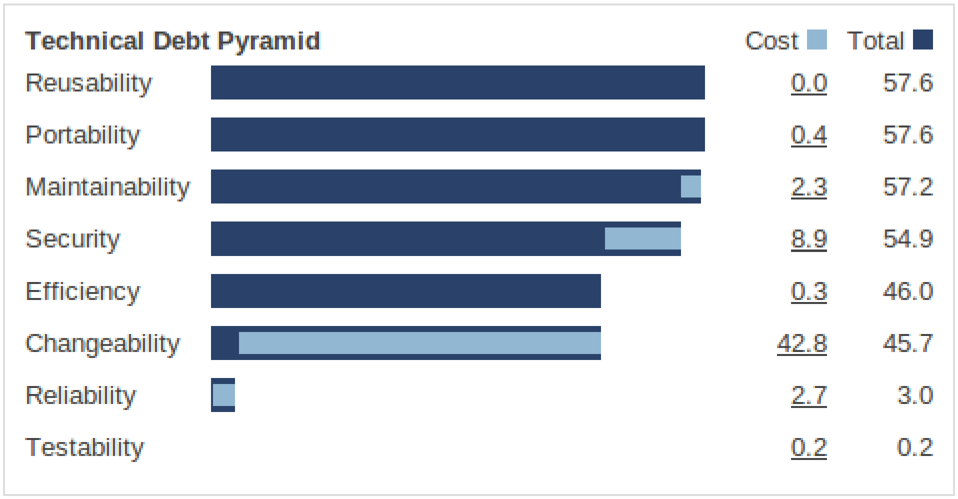
```
    } catch (Throwable t) {
```

Throwable and Error classes should not be caught | Open | Updated: 22 days |
Author: simon.brandhof@gmail.com | Technical debt: 20 minutes

Catch Exception instead of Throwable.

The technical debt for each issue is set at the rule level. If you've got the commercial SQALE plugin, you can adjust the estimate for each rule (in doing so, you're editing the SQALE Analysis Model – the remediation cost of each rule). But those estimates were made by seasoned professionals, so you probably won't need to.

So now you know how long it will take to fix the application, but how do you prioritize the work? There's a widget for that. It's called the technical debt pyramid, and it looks like no pyramid you've ever seen before:



Technical Debt Pyramid

	Cost	Total
Reusability	0.0	57.6
Portability	0.4	57.6
Maintainability	2.3	57.2
Security	8.9	54.9
Efficiency	0.3	46.0
Changeability	42.8	45.7
Reliability	2.7	3.0
Testability	0.2	0.2

Don't be confused by the fact that this doesn't look like something from ancient Egypt; this is a figurative pyramid. The way to read it is from the bottom up. The bottom row will always have the smallest bar in the bar graph, but it has the biggest import – because it’s foundational. The rows in this widget each represent a “characteristic” and each characteristic builds on the ones below it. Testability is at the bottom because it’s most important: first you make sure your app is testable, then you make sure it’s reliable, then changeable, and efficient, and so on.

The bars in the graph show remediation time per characteristic. The light blue portion shows the time to clean up *this* characteristic, and the dark blue portion shows cumulative time, working from the bottom up. As usual, each portion of the widget clicks through to a drilldown to let you see exactly where the technical debt for a characteristic is.

Going Further

The SonarSource SQALE plugin extends the technical debt feature embedded in SonarQube. Among other features, it allows to tune the SQALE model (adjust the remediation estimates for each rule, set the list and order of characteristics, change which characteristic a rule falls under, and more), provides additional widgets, etc.