Debt Accounting on Metrix

WHITEPAPER

SQALE for ALM artefacts — computing, summing and tracking engineering debt across requirements, tests and code.

T4Test · November 2026

Debt Accounting on Metrix

SQALE for ALM artefacts: computing, summing and tracking engineering debt across requirements, tests and code

Published by T4Test Engineering Consulting · November 2026 **For:** process engineers, quality leads, functional-safety managers **Companion to:** *The Metrix Filter Language, Compared*

Abstract

Code analysers already compute technical debt. SonarQube, CAST and tools that implement the SQALE method sum remediation hours across code-rule violations and produce one number in person-days. What those tools cannot see is the other half of the engineering debt in a real programme: unlinked requirements, tests without a verdict, reviews that never completed, ASIL levels that do not agree along a chain, agreements between two parties that are still open. ALM tools, which could see those, do not sum. Nobody produces one number across both.

This paper shows how to do it with Metrix today. The reused model is SQALE: one remediation function per rule, summed across the result set. Metrix supplies the rest: a filter language that reads across tools, scopes that name the universe, derived attributes that carry the per- item remediation effort and a per-item confidence in that estimate, and a sum aggregate that produces the principal in person-days — with a trend and a drill-down for free. Each debt figure is reported with a planning band around it: wide when confidence is low, narrow when it is high. The band tightens as the team closes items and learns what each rule family actually costs, without the point estimate drifting for the wrong reasons. No DSL extension is needed. One rule file in the repository plus a few derived attributes turn SQALE into a running debt register for the whole programme.

We demonstrate the method on a constructed cross-tool example — requirements in Polarion, tests in Jira Xray, code in git — and compute debt in person-days, split by category, with a one-sprint trend. The example is small on purpose. The point is not the number. The point is that the same rule set produces a number across all three tools, and that the sum can be trusted because every term in it is traceable back to the rule that owed it.

1. The problem

Every engineering programme owes two kinds of debt.

Code debt is the kind SQALE-style tools already measure. Rule violations in source files. Each rule carries a remediation function. Sum the hours, report one number. SonarQube does this out of the box. CAST does it in currency. CISQ (ISO/IEC 5055) standardises the method. A mid-sized codebase typically carries

somewhere between 50 and 500 person-days of code debt, and the number is actionable because it attaches to files.

Process debt is what the rest of the programme owes. A requirement with no decomposition. A test that ran but the verdict field is blank. A review that was opened and never closed. An agreement between system and software engineering that is still open after six sprints. A decomposition from ASIL B into two children both marked QM, with no independence record. Each of these is an outstanding obligation. Each costs effort to repay. None of them are visible to a code analyser.

ALM tools surface the symptoms — a dashboard tile that says "*14 requirements without verification*" — but they do not sum. The dashboard reports a count. It does not say "*the open items on this list represent 86 person-days of work, 52 of which block the next baseline.*" An engineer looking at the tile cannot plan against it.

The gap is not that a summing model is missing. SQALE has existed since 2012 and is well- established for code. The gap is that **the model has never been applied across the tool boundary**. Code analysers do not reach the ALM. ALM tools do not sum.

Metrix does both.

---

2. The reused model: SQALE in one page

SQALE (Software Quality Assessment based on Lifecycle Expectations) is a method for measuring technical debt. It is built from three parts:

- **A rule set.** Every quality rule the organisation cares about is listed. For code, this is usually a vendor catalogue (SonarQube ships about 600).
- **A remediation function per rule.** The function maps a violation to an effort estimate — usually in minutes or hours. SQALE defines four shapes:
- **constant** — e.g., 5 min to delete an unused import;
- **linear per occurrence** — e.g., 2 min per duplicated name;
- **linear per size** — e.g., 2 min per 10 duplicated lines;
- **polynomial** — for refactorings that scale badly with complexity.
- **A summing step.** The total debt is the sum of remediation efforts across every violation in the codebase. The result is a number in person-days.

Two derived figures round it off. The **Technical Debt Ratio** = debt / cost-to-rebuild-from- scratch; it normalises the absolute number so that a 50 kLOC project with 20 days of debt is not directly compared to a 500 kLOC project with 200 days. And the **SQALE rating** = A to E, bands on the ratio.

SQALE is deployed everywhere. SonarQube uses it. CAST uses a variant. CISQ (the Consortium for IT Software Quality) wrapped it into ISO/IEC 5055. For code debt it is the accepted method, and this paper does not re-derive it. What we do is lift it off the code and apply it to the process artefacts the same programme owes debt on.

---

3. The Metrix primitives used

No DSL extension is proposed. Every mechanism below ships in Metrix v1.0 or lands in v1.1.

What we need	Metrix primitive	Released in
A universe of items the rule applies to	**Scope**	v1.0
A short predicate on one item (the rule itself)	The DSL filter	v1.0
A hop with a filter on the far end	`Rel links Scope { … }`	v1.0
A chain across tools	Scopes over different connectors in one graph	v1.0
A remediation function per rule	**Derived attribute**	v1.1
Sum remediation across a result set	`sum('remediationEffort')`	v1.0
Breakdown per owner or category	**Evaluator**	v1.0
Trend across sprints	**Trend sidecar**	v1.0
Rule identity across runs	**Query fingerprint**	v1.0

A rule in Metrix is a predicate: an item either satisfies it or it does not. Per-item remediation effort is summed over the items that do not. The three categories of debt below are a useful way to split the violated items by *why* they failed and *who repays them*, not three different verdicts:

- **Data debt** — an attribute the rule expects was never set. Repaid by one field entry. Usually cheap, and worth repaying first because it often reveals which bucket the item really belonged to.
- **Decision debt** — a decision has to be made and recorded (an exception to justify, a decomposition to declare, an owner to assign). No new artefact; just a statement.
- **Artefact debt** — a missing link, a missing test case, a missing review record. Needs engineering work, typically the most expensive.

Each category is a small predicate in its own right (`unspecified('safetyClass')`, `not specified('Safety-assessment reviewed')`, etc.), so each becomes its own sibling metric if you want the split in the report.

---

4. The mapping

A rule in this paper has five parts, all expressible in Metrix today:

Rule part	Metrix construct
Scope — which items the rule applies to	a Metrix scope
Predicate — what satisfies the rule	a DSL filter, including hops
Exception — items that look violated but are not	a boolean attribute carried in the predicate
Remediation cost — effort to bring one item to satisfied	a derived attribute `remediationEffort`
Confidence — how certain the cost estimate is	a derived attribute `remediationConfidence` in [0, 1]

Two metrics are produced per rule — the one that satisfies it, and the complement:

```
// applied to the 'Sw Req safety-related' scope

m_satisfied = VerifiedBy links 'Test Case' { 'Verdict' == "Passed" }
              or ( 'Derived' == true and 'Safety-assessment reviewed' == true )

m_violated  = not m_satisfied
```

Three figures are reported together: the point estimate (the best guess) and a band around it that widens when confidence is low:

```
// over the result set of m_violated

debt_hours_estimate = sum('remediationEffort')
debt_hours_low      = sum('remediationEffort' *  'remediationConfidence')
debt_hours_high     = sum('remediationEffort' * (2 - 'remediationConfidence'))
```

At confidence = 1 the band collapses onto the point estimate. At confidence = 0.5 each item's band is ±50 % of its estimate. At confidence = 0.3 the band is ±70 %. The **estimate** is the number the sprint plan runs against; the **band** is the budget for being wrong. A programme with a 100 h estimate and a [60 h, 140 h] band is in a different conversation from one with the same 100 h estimate and a [95 h, 105 h] band, even though the headline number is identical.

The gap between `low` and `high` is itself a useful signal: a wide gap says "*we have not closed enough items of this kind to know yet*", and the next sprint repays the rule family that most tightens the band per hour invested.

An evaluator breaks either figure down by category (data / decision / artefact), by owner or by sprint-of-origin. The trend sidecar tracks both across baselines. The drill-down names the item that owes each hour.

4.1 Confidence tightens over time

Confidence starts low — typically 0.3 for a brand-new rule family — and tightens as evidence accumulates. Three mechanisms, all driven by data Metrix already collects:

- **Author override.** An engineer who knows the category well raises or lowers the confidence on a specific item, with a one-line justification. This is the explicit knob: "*yes, this one really is 2 h, I just*

finished one like it last week; confidence 0.9."

- **Repayment learning.** When a violated item is repaid, Metrix records the actual effort spent (from the commit timestamp, the time-tracking tool, or the engineer's own entry). After a handful of closures in a category, the derived attribute updates its central estimate and its spread. The next sprint's `remediationEffort` is drawn from the measured distribution rather than the workshop guess.
- **Age decay in the opposite direction.** An item still open two quarters after it was created has quietly drifted from its original estimate: the surrounding code has moved, the author has moved, the scope has shifted. Confidence in the original number decays with age, which is a signal to re-estimate rather than a signal to ignore.

Each mechanism is a small addition to the derived attribute's expression. The engineer never touches the DSL to use them; the rule family's maintainer wires them in once, and the accounting tightens itself from there.

---

5. A worked example

The example is constructed, not drawn from a real customer. It is minimal on purpose.

The programme. A hypothetical driver-assistance feature (adaptive high beam) under ISO 26262. Three tools:

- Requirements in Polarion.
- Test cases and verdicts in Jira with Xray.
- Code in GitHub under `/src/adaptive-high-beam/`.

The rule. Every safety-related software requirement must trace to at least one test case with a passing verdict. Safety class (QM / A / B / C / D) is carried on every requirement. Derived requirements are an accepted exception if the safety assessment has been reviewed.

The seed. 48 safety-related SW requirements, 71 test cases, 13 500 lines of Kotlin. Of the 48 requirements, 12 have an unset safety class; 6 are declared derived (3 with a review reference, 3 without).

5.1 The rule in Metrix

Written once, kept in the repo alongside the Metrix book. The scope attaches the predicate to safety-related software requirements:

```
 // scope: 'Sw Req safety-related' — Polarion SW requirements with safety == true

m_satisfied = VerifiedBy links 'Test Case' { 'Verdict' == "Passed" }
              or ( 'Derived' == true and 'Safety-assessment reviewed' == true )

m_violated  = not m_satisfied

// derived attributes, computed per item on the 'Sw Req safety-related' scope:
//   remediationEffort     = 8h  if safetyClass in (C, D) else 4h
//   remediationConfidence = 0.4 at creation; 0.8 after 5 similar items were closed;
//                           decay back towards 0.4 after 90 days still open
```

The remediation cost follows SQALE: constant per rule, with a step-up for high-safety items. The confidence starts at 0.4 (the team has not yet closed any items of this kind) and updates with experience. Derived attributes (Metrix v1.1) compute both per item.

5.2 Running the rule

Metrix evaluates the rule over the three-tool graph. 24 of the 48 safety-related software requirements satisfy the predicate; 24 violate it. The sum of `remediationEffort` is the point estimate; applying the confidence gives the planning band around it:

Category of violation	Items	Estimate	Confidence (avg)	Band (low–high)
Missing test link	3	12 h	0.80	9.6 – 14.4 h
Missing test (not written)	6	24 h	0.40	9.6 – 38.4 h
Declared derived, no safety assessment	3	6 h	0.70	4.2 – 7.8 h
Safety class never set — the predicate cannot be evaluated	12	2 h	0.95	1.9 – 2.1 h
Total principal	**24**	**44 h**		**25.3 – 62.7 h**

One headline estimate and a band: **44 h, [25 h, 63 h]**. A link-pair coverage measure on the same graph would report 24 / 48 = 50 % satisfied and stop there. The breakdown by category, the sum in person-days and the confidence band are what the Metrix pipeline adds.

The confidence column already reflects team knowledge: missing-link items are familiar (the fix is mechanical), so the 0.80 is high and the band tight; missing-test items vary wildly in scope, so the 0.40 signals genuine uncertainty and the band is ±60 %; the safety-class entries are almost certain (10 minutes each unless the engineer is on holiday), so the band is ±5 %. The 44 h is what the sprint plan runs against. The gap to 63 h at the top of the band is the budget for being wrong.

5.3 Debt by category

Collapsing the breakdown above into the three repayment categories:

Category	Repayment action	Items	Debt hours
Data	Set the `safetyClass` attribute	12	2 h (10 min each)
Decision	Open and attach the safety assessment	3	6 h (2 h each)
Artefact	Write and link the missing test, or author the missing requirement	9	36 h

Total principal: **44 person-hours**. The data debt is cheap — one afternoon of field entry — and worth repaying first, because 12 of those items might turn out to need a test (artefact debt) or simply fall outside the scope once the class is known. The sum is what the sprint plan runs against.

5.4 Per owner, per sprint

Evaluators naturally break the figure down further:

- **By owner** (system engineer / software engineer / verification lead): each gets their own backlog, in hours, not a percentage.
- **By sprint-of-origin** — how long each item has been outstanding. The trend sidecar captures this automatically, as long as the rule has not changed (and the fingerprint tells us when it does).

5.5 Trend after one sprint

The team repays the data debt and half of the artefact debt. Two things move: the point estimate drops, and the band tightens as the team learns what the items in each category actually cost:

Sprint	Estimate	Band	Confidence (avg)
N	44 h	25.3 – 62.7 h	0.57
N+1	20 h	15.0 – 25.0 h	0.75

The estimate dropped from 44 h to 20 h. More importantly the band tightened from ±85 % around the midpoint to ±25 %: closing five missing-test items taught the team that *this* kind of test typically takes 3 h, not the 4 h in the original guess, and that the spread around that number is small. The chart shows both. The drill-down lists which specific items were closed, by whom, and what they actually took. The next planning conversation is a conversation about 20 hours give or take 5, not about a percentage that moved for reasons no one can fully explain.

---

6. What this adds to the existing tools

Capability	SonarQube / CAST	ALM dashboards	Metrix (with this method)
Sum debt in person-days	✓ (code only)	—	✓ (code + ALM)
Reads across tool boundaries	—	partial	✓
Declared rules, not vendor-shipped	—	—	✓
Splits debt by category (data / decision / artefact)	—	—	✓
Planning band around the estimate, tightening with experience	—	—	✓
Rule identity over time (fingerprint)	—	—	✓
Trend out of the box	✓	partial	✓
Per-owner breakdown	partial	✓	✓

The row that matters most is the first: **this is the only method that sums both halves of the debt a programme actually carries**. The others are not wrong. They are each a part of the picture.

---

7. Limitations (honest ledger)

- **Remediation functions need elicitation.** SQALE's code-rule effort catalogue does not transfer to ALM items. An organisation adopting this method writes its own `remediationEffort` derivations — typically a one-day workshop per rule family. Expect the first estimates to be rough and to settle after two or three sprints of use.
- **Validity is assumed, not shown.** An attribute set wrongly, or a link between items that do not correspond, produces a verdict that is numerically fine and factually wrong. Review effort is debt of its own kind and is not accounted for in the sum.
- **Proof, not survey.** The example in Section 5 is constructed. It demonstrates that the method composes and produces usable numbers. It does not show how often such debt occurs in the field. That is a longitudinal study, and it is the natural next piece of work.

---

8. Related work

- **Letouzey, 2012** — *The SQALE method*. The summing model and the remediation function.
- **CISQ / ISO/IEC 5055** — the industry standardisation of automated technical-debt measurement; the method SonarQube and CAST both report against.
- **Rempel & Mäder, 2017** — *Preventing defects: the impact of requirements traceability completeness on software quality*. Evidence that trace completeness affects defect rate.

- **The Metrix Filter Language, Compared (T4Test, 2026)** — the companion whitepaper. Explains the DSL constructs used here against SQL, JQL, WIQL, Polarion Lucene and other query languages.

---

9. Conclusion

Debt accounting is not a new idea. Summing it is not a new idea. What has been missing is a tool in a position to apply the model across the whole programme — not just the code, not just the ALM, but both at once, with one declared vocabulary and one trend.

Metrix is in that position today, with the primitives it already ships. A short rule file in the repository, a few derived attributes for the remediation cost and the confidence in it, and the team has a debt figure it can plan against. The point estimate is the headline; a planning band around it says how sure the team is in that number. The band tightens as the team closes items and learns what each rule family actually costs. The category split (data / decision / artefact) tells the organisation which part of the figure is one afternoon of field entry away and which part is real engineering.

The invitation is small on purpose. Three or four early-adopter teams, one rule family each, one quarter of measured trend. That is enough to turn a well-founded method into a running practice. The organisations willing to try it get a number nobody in the market can give them today.

About T4Test

T4Test Engineering Consulting is an independent engineering consultancy for critical-systems development in Europe. We build PROmeter — a language and tool family for engineering data, designed for teams that work under ISO 26262, IEC 62304, DO-178C and similar regimes.

To discuss early adoption, write to contact@t4test.de or visit https://t4test.de.

© 2026 T4Test Engineering Consulting. All rights reserved.