Risk Architecture

Part 1 — Scoring Logic for Individual Measures

Version 49.1 · 30 August 2026

Reconstructed from the code, measure by measure.

This edition replaces *Risk Architecture Part 1, Scoring Logic for Individual Measures* (June 2026). It is not an update of that document. Every entry here was rebuilt by reading the analyzer that implements it, because a documentation audit in August 2026 found that the previous edition and the code had diverged in 112 places, and that where they disagreed the code was sometimes right and sometimes wrong.

The rule applied throughout: **this document describes what the tool does.** Where a measure behaves in a way that looks unintended, the behaviour is documented and the concern is recorded in the defects appendix rather than quietly corrected in the prose. A reference that disagrees with the tool is worse than one that admits what the tool does.

Contents

1. How the framework works

The risk architecture runs in three scoring layers, preceded by an eligibility step added in August 2026.

Eligibility. Before anything is scored, the tool establishes whether an institution's financial position is independently measurable. An institution whose finances are reported by a parent — the University of California campuses, the Pittsburgh regionals, Rutgers Newark and Camden — or which files no IPEDS finance at all is reported in full but is not ranked, and carries a stated reason. This is described in *Part 3, Reading Risk Assessment Tool Output*.

Layer 1 — individual measures. Each measure scores risk points against its own maximum, from fixed thresholds, from peer position, or from both. **This document describes Layer 1.**

Layer 2 — measure combinations. Archetypes, patterns and accelerators score combinations that no single ratio captures, adding up to 50 points. Described in *Part 2*.

Layer 3 — assignment. Total points over the applied maximum, mapped to a band: Low 0–25%, Moderate 26–44%, High 44–68%, Critical above 68%.

Two properties of Layer 1 that govern everything below

The denominator is not fixed. A measure returning `not_applicable`, `Unknown` or `insufficient_data` leaves *both* the numerator and the denominator. It does not score zero; it leaves the framework. An institution's maximum therefore depends on how much data it has, and two institutions' composites are not comparable unless their coverage is.

Data gaps must never score. Where a measure cannot form a value, it must contribute no points and, where the whole measure is unavailable, must set `isDataUnavailable` so the orchestrator removes it from the denominator. Several measures violated this before August 2026 — a missing enrolment series scored 3 points, a missing Pell figure scored 1, and salaries and benefits charged a point each for an absent growth trend and an absent peer comparison. Those are fixed. The remaining violations are listed in the appendix.

2. How to read an entry

Each measure is documented under the name it carries in tool exports. Entries state the implementing class, the data file the measure reads, what it measures, how the sectors differ, the actual scoring thresholds, the maximum and whether that maximum is reachable, how the risk level is derived, what happens when data is missing, and how the peer group is formed.

Thresholds are given as the code applies them, and each is marked as a **floor** (value ≥ X) or a **ceiling** (value ≤ X). This matters more than it sounds: several measures were found in August 2026 to have their comparison written one way and documented the other.

Where a change was made in August 2026, the entry records it and quotes the reasoning from the code, because those comments are the only surviving statement of design intent for several measures.

Where something could not be determined from the code, the entry says **UNCLEAR** and names what is unresolved. Nothing in this document is inferred.

3. The framework

Maximum risk points by measure and sector, from `risk_scoring_engine.js`. A dash means the measure is not present in that sector's framework at all; a zero means it is present, runs, and is reported as **Informational** without contributing points.

Sector totals: **public 217, private 239, private for-profit 189, community college 184.** The for-profit framework has never been run.

Measure	Public	Private	For-profit	CC	Category
150% Graduation Rate	8	8	8	8	Student Success
Academic Support Expenditure	—	—	—	3	Cost Management
Adjusted Operating Margin	8	8	8	8	Financial Performance
Brand Pipeline - Acceptance & Yield	10	8	8	0	Market Position
Capitalization Ratio	10	10	10	10	Financial Reserves
Debt-to-Revenue Ratio	8	10	10	10	Financial Leverage

Measure	Public	Private	For-profit	CC	Category
Endowment Dependency Ratio	0	6	—	0	Revenue Diversification
Endowment Draw Sustainability	0	10	—	0	Financial Reserves
Endowments per FTE	6	12	—	0	Financial Reserves
Enrollment Trends	15	18	10	15	Market Position
Estimated Impacts of Changing Federal Policy Landscape	0	0	0	0	Federal Policy Risk
Institutional Aid	6	2	2	0	Revenue Diversification
Institutional Support	6	6	6	6	Cost Management
Instructional Support	6	6	6	6	Cost Management
Net Asset Restriction Profile	0	6	—	0	Net Asset Composition
Net Revenue Ratio	0	0	0	0	Financial Performance
Net Tuition per FTE	6	6	6	6	Pricing Strategy
Operating Margin	0	0	0	0	Financial Performance
Pell Grant Recipients	4	4	4	4	Student Demographics
Primary Reserve Ratio	10	10	10	10	Financial Reserves
Program Distribution	6	6	6	6	Academic Quality
Programs Sustainability	12	12	12	12	Academic Sustainability
Research Expenditure	—	—	—	0	N/A
Research Support	8	8	—	0	Revenue Diversification
Return on Net Assets	10	10	10	10	Financial Performance
Risk Adjustment (Layer 2)	0	0	0	0	Risk Adjustments
Salaries & Benefits	6	6	6	6	Cost Management
State Appropriations Dependency	15	0	—	15	Financial Stability
Structural Operating Margin	8	8	8	8	Financial Performance
Student Retention	6	6	6	6	Academic Quality
Student Support	6	6	6	6	Cost Management
Student-Facing Expenditures	6	6	6	6	Cost Management
Student-Faculty Ratio	2	2	2	2	Academic Quality
Transfer Out Rate	8	8	8	0	Student Retention
Tuition Dependency	8	18	18	8	Revenue Diversification
Unrestricted Net Assets per Student FTE	5	5	5	5	Financial Reserves
Viability Ratio	8	8	8	8	Financial Stability

Thirty-seven rows, thirty-four entries in §4. Three rows are not documented as measures: **Risk Adjustment** is the Layer 2 engine and carries a framework maximum of zero in all four sectors, so it appears here only because it is registered like an analyzer; it is described in *Part 2*. **Academic Support Expenditure** (community college, 3 points) and **Research Expenditure** (community college, 0 points) run only in the community college framework and were not reached in the August 2026 reading of the code. They are listed in the appendix as undocumented.

4. The measures

34 measures, alphabetical by the name each carries in tool exports.

150% Graduation Rate

`getAnalyzerDisplayName` has no entry — publishes as `150percentgradrate`

- **Class**: `Unified150PercentGradRateAnalyzer` (wrapper: `Universal150PercentGradRateAnalyzer`, wrapper label `'150% Graduation Rate Analysis'`, analyzer key `150percentgradrate`)
- **Data key**: The analyzer does not call `getDataForAnalyzer` itself. The wrapper (line 2533) calls `this.dataProcessor.getDataForAnalyzer('150_percent_grad_rate')`, which maps to `/annualized.*150.*percent.*time.*pv.*(and|&).*pu/i` (`data_processing_pipeline.js` line 44).
- **What it measures**: The percentage of students completing within 150% of normal time, read from bare four-digit year columns 2010–2024. The file header states "Higher graduation rates = lower risk … Decreasing trends = increasing risk". Scored on the latest year's rate and on the first-to-last percent change.
- **Sector differences**: **None.** No threshold, point value or maximum varies by sector. `sectorName` is used only for peer-pool filtering (`extractAllInstitutions()` keeps rows whose SECTOR equals `this.sector`) and for narrative text. The orchestrator's `CC_EXCLUDED_ANALYZERS` list no longer excludes this analyzer for sector `'4'` — the 2026-08-09 comment (lines 48–54) records that the original 100%-Unknown finding traced to the workbook carrying `GBA6RTT` rather than `GRRTTOT`, and states "CC completion is now measured on transfer-out AND 150% completion, at parity". However the HTML display filter `CC_EXCLUDED_KEYS` (line 3202–3206) still contains `'150percentgradrate'`, so for sector `'4'` the analyzer runs and scores but its row is filtered out of the results display. See Notes.

Scoring

Level is `max(absolute leg, peer percentile leg)`; trend is independent and additive.

Level, absolute leg (`assessMeasureRisk`, lines 604–614). Ceilings, evaluated in order; nothing is scored at 30% or above:

Condition	Points
`currentValue < 15.0`	5
`currentValue < 20.0` (and ≥ 15)	4
`currentValue < 25.0` (and ≥ 20)	3
`currentValue < 30.0` (and ≥ 25)	2
`currentValue >= 30.0`, or null	0

Level, peer percentile leg (`gradLevelFromPercentile`, lines 574–582). Ceilings:

Condition	Points
`percentile < 10`	4
`percentile < 25` (and ≥ 10)	2.5
`percentile < 50` (and ≥ 25)	1
`percentile >= 50`, or null/non-finite	0

Level contribution = `Math.max(absolutePoints, percentilePoints)` → 0–5.

Trend leg (lines 625–635). Only applies when `trends.direction === 'decreasing'`, which itself requires `overallChange < -3` percentage points. `percentChange` is the relative percent change and defaults to 0 when null. Ceilings:

Condition (given `direction === 'decreasing'`)	Points
`percentChange < -25`	3
`percentChange < -10` (and ≥ −25)	2
`percentChange >= -10`	1
`direction !== 'decreasing'`	0

Total = level + trend, then `Math.min(total, 8)`.

- **Maximum points**: `this.measureConfig.maxRiskPoints = 8` (line 50), published as `analysisResults.maxRiskPoints`. The wrapper re-asserts 8 on the success path (line 2560). 5 + 3 = 8, so the cap is reachable.
- **Risk level mapping**: `assessMeasureRisk` lines 641–643, on the capped total:

Condition	riskLevel
`riskPoints >= 6`	`High`
`riskPoints >= 3` and `< 6`	`Moderate`
`riskPoints < 3`	`Low`

The inline comments label these "75%+ of max points" and "~40%+ of max points".

- **Data-gap behaviour**:
  - No unitId → `createErrorResult('MISSING_UNITID', …)`: `riskPoints: 0`, `maxRiskPoints: 8`, `riskLevel: 'Unknown'`, `isDataUnavailable: true`.
  - Institution not found, or zero usable year values → `createFallbackResult()`: `riskPoints: 0`, `maxRiskPoints: 8`, `riskLevel: 'Unknown'`, `isDataUnavailable: true`.
  - Thrown error → `createErrorResult('ANALYSIS_ERROR', …)`, same shape.
  - Wrapper paths (institution missing, data file not loaded, thrown error) all return `riskPoints: 0`, `maxRiskPoints: 8`, `riskLevel: 'Unknown'`, `isDataUnavailable: true`.
  - With exactly one usable year, `dataAvailable` is true, the absolute leg scores, and the trend leg is 0 (`direction: 'insufficient-data'`).
- **Peer logic**: Same construction as retention — same-sector rows from `extractAllInstitutions()`, minus self, filtered by `carnegieBasic` array membership, `hbcuFilter`, and `customPeerGroup`. Peer series are parsed through a `WeakMap` row memo (`__memoRow`, 2026-08-26). The scoring comparison is a **percentile**: `benchmarkValue()` computes `round(100 * count(peers with LOWER grad rate) / peer count)` and that number feeds `gradLevelFromPercentile`. `calculateStatistics()` also produces mean, median, min and max, but only the mean is used, in narrative findings.
- **Notes**:
  - `getAnalyzerDisplayName()` has **no key** for `'150percentgradrate'`. The fallback branch (`key.replace(/_/g,' ').replace(/analyzer/g,'').trim().replace(/\b\w/g, …)`) leaves the string as the literal `150percentgradrate`, because the first character is a digit. That is the label the enhanced export prints.
  - `measureConfig.riskThresholds` (`excellent:70, adequate:50, moderate:35, critical:20`) is not used by the scorer. Its values are duplicated as literals inside `assessCompletionStatus()`, which only produces the `completionStatus` string.
  - `measureConfig.riskWeights` (`currentRate 0.5, trend 0.3, peerComparison 0.2`) is declared and never read.
  - The comment at lines 592–602 records the removed gate: "rate < 30% AND trend !== 'increasing'", which produced "two independent ways to score zero"; it cites 86% and 43% both scoring 0.
  - The comment at lines 571–573 says the percentile leg is "Capped below the absolute leg's maximum"; the values are 4 versus 5, consistent with that claim.
  - `extractMeasureData()` drops any parsed value `< 0`.
  - The orchestrator's `RA_CANONICAL` map has no entry for `150percentgradrate`; the canonical measure name falls through to `result.measure`, i.e. `'150_percent_grad_rate'`. But the wrapper strips `measure` from the result (see Shared wrapper behaviour), so the fallback is `analyzerKey`.
  - 2026-08-26 comments at lines 314–375 document the `__unitRow` first-wins index and the `__memoRow` WeakMap peer-series cache as pure performance changes.

Adjusted Operating Margin

OPERATING PERFORMANCE & COST STRUCTURE

- **Class**: `UnifiedAdjustedOperatingMarginAnalyzer`
- **Data key**: `'adjusted_operating_margin'` (wrapper `UniversalAdjustedOperatingMarginAnalyzer`, HTML line 2397). A **second** key is read at scoring time by `getOMDataForComparison`: `window.dataProcessor.getDataForAnalyzer('operating_margin')`, used only by the divergence kicker. Internal `this.measureName = 'operating_margin'` — **not** `'adjusted_operating_margin'`; it is the identical string used by `UnifiedOperatingMarginAnalyzer`. Wrapper label string is `'Adjusted Operating Margin Analysis'`. Dataset regex (`data_processing_pipeline.js` line 32): `/annualized.*adjusted.*operating.*margin/i`. Declared `dataRequirements`: `['annualized adjusted operating margin pu.xls']`.
- **What it measures**: The file is a copy of the Operating Margin analyzer pointed at the adjusted series; its header comment and narrative text are unchanged from the parent ("Operating margin measures the percentage of revenue remaining after operating expenses…"). The measure is read from pre-computed year columns 2010–2024, not computed. The reason the adjusted measure exists is stated in the divergence-kicker comment (2026-08-26): "Under GASB, state appropriations are NONOPERATING revenue, so a public university's unadjusted operating margin is negative by construction - that is the entire reason the adjusted measure exists."
- **Sector differences**:
  - `sectorName`: `'1'` → Public, `'2'` → Private, `'3'` → For-Profit, anything else → Community College.
  - **`maxPoints` is 8 for every sector**; there is no sector switch on the maximum.
  - **Factors 1 and 3** each branch two ways: `sector === '1'` (public) versus **everything else**. Sectors `'2'`, `'3'` and `'4'` all take the "private university" branch. Community colleges are scored on private thresholds; no comment addresses this.
  - **Factor 5 (divergence kicker) is private-only**: `if (this.sector !== '2') return 0;`.
  - Framework points: 8 in all four sectors (`public`, `private`, `privateForProfit`, `communityCollege`), confirmed in both `risk_scoring_engine.js` and `framework.json`. The HTML wrapper additionally forces `result.analysisResults.maxRiskPoints = 8` with the comment "Both sectors get 8 points".
- **Scoring**:

`maxPoints = this.measureConfig.maxRiskPoints = 8`. `riskPoints` is clamped with `Math.min(riskPoints, maxPoints)` after all five factors, so the raw sum can exceed the cap (raw maximum reachable is 5 + 2 + 5 + 2 + 1 = **15**, clamped to 8).

Factor 1 — 3-year rolling average. Ceiling tests, first match wins.

Sector `'1'` (Public)	Direction	Points	Factor string
`rollingAverage < -2.0`	ceiling	5	Critical 3-year average operating margin for public university (<-2%)
`< 0.0`	ceiling	4	Negative 3-year average operating margin for public university (0% to -2%)
`< 2.0`	ceiling	2	Weak 3-year average operating margin for public university (0-2%)
`< 4.0`	ceiling	1	Fair 3-year average operating margin for public university (2-4%)
`>= 4.0`	floor	0	—

All other sectors (`'2'`, `'3'`, `'4'`)	Direction	Points	Factor string
`rollingAverage < -15.0`	ceiling	5	Critical 3-year average operating margin for private university (<-15%)
`< -10.0`	ceiling	4	Severe 3-year average operating margin for private university (-15% to -10%)
`< -5.0`	ceiling	3	Weak 3-year average operating margin for private university (-10% to -5%)
`< 0.0`	ceiling	2	Fair 3-year average operating margin for private university (-5% to 0%)
`< 3.0`	ceiling	1	Good 3-year average operating margin for private university (0-3%)
`>= 3.0`	floor	0	code comment: "0 points if >= 3%"

Factor 2 — peer comparison (max 2 points). Identical to Operating Margin; guarded by `benchmarking.currentPerformance && rollingAverage !== null`.

Condition	Points	Factor string
`quartile === 'Bottom quartile'` (percentile < 25)	2	Operating margin significantly below peer institutions
`quartile === 'Below median'` (25 ≤ percentile < 50)	1	Operating margin below peer institution median
otherwise	0	—

Factor 3 — current-year value. Ceiling tests, first match wins.

Sector `'1'` (Public)	Direction	Points	Factor string
`currentValue < -10.0`	ceiling	5	…extreme financial distress (<-10%)
`< -5.0`	ceiling	4	…severe financial distress (-10% to -5%)
`< -2.0`	ceiling	3	…significant financial stress (-5% to -2%)
`< 0.0`	ceiling	2	…emerging financial concerns (negative)
`< 2.0`	ceiling	1	…minor concerns (0-2%)
`>= 2.0`	floor	0	—

All other sectors	Direction	Points	Factor string
`currentValue < -20.0`	ceiling	5	…extreme financial distress (<-20%)
`< -15.0`	ceiling	4	…severe financial distress (-20% to -15%)
`< -10.0`	ceiling	3	…significant financial distress (-15% to -10%)
`< -5.0`	ceiling	2	…moderate financial stress (-10% to -5%)
`< 0.0`	ceiling	1	…minor financial concerns (negative but manageable)
`>= 0.0`	floor	0	code comment: "0 points if >= 0%"

Factor 4 — trend deterioration (max 2 points). LIVE in this file since 2026-08-28. Applies only when `measureAnalysis.trends?.direction === 'declining'` (which itself requires `overallChange < -1`, i.e. last-minus-first below -1 percentage point).

Condition on `annualizedChange`	Direction	Points	Factor string
`< -2.0`	ceiling	2	Operating margin declining rapidly (>2% annually)
`< -1.0` (and ≥ -2.0)	ceiling	1	Operating margin declining moderately (1-2% annually)
`>= -1.0`	floor	0	—

The 2026-08-28 (v49.1) comment above this block is authoritative:

> *"read `annualizedChange` instead of `changeRate`. WAS WRONG: this factor read `measureAnalysis.trends?.changeRate`, but `calculateMeasureTrends` never writes a `changeRate` key — it returns direction, overallChange, rollingTrend, annualizedChange, dataPoints and timespan. `changeRate` was therefore always undefined, `|| 0` coerced it to 0, and neither 0 < -2.0 nor 0 < -1.0 can hold: the factor has been INERT since it was written, awarding 0 of its 2 points to every institution. EVIDENCE: replaying this analyzer's own extraction and trend maths over the source file, roughly 100 institutions gain points once the correct key is read. Eight layer-2 gates consume this analyzer's output, so this propagates. UNITS: extractMeasureData normalises every row to PERCENTAGE POINTS via rowScale/parseScaled (decimal rows are multiplied by 100), so overallChange and hence annualizedChange = overallChange / yearsDiff are percentage points per year — the same units as the -2.0 / -1.0 thresholds. No rescaling required. NOW: a genuinely declining margin scores 2 points (>2pp/yr) or 1 point (1-2pp/yr)."*

UNCLEAR: the units claim in that comment. It states that `extractMeasureData` "normalises every row to PERCENTAGE POINTS via rowScale/parseScaled (decimal rows are multiplied by 100)". `extractMeasureData` (line 525) does call `this.rowScale(row)` and `this.parseScaled(...)`. But the block comment on `rowScale/parseScaled` at line 1097, dated the same period, is headed "**rowScale / parseScaled — RETAINED, NOT USED FOR SCORING. Reverted 2026-08-26**" and says "Per-row unit detection was applied here and then withdrawn the same day … Restored to the original rule until then." The code as it stands does call them from `extractMeasureData`; the header comment says they are not used. The two cannot both be true, and this extraction does not resolve which is intended.

Factor 5 — divergence kicker (+1, private only). `calculateDivergenceKicker` returns 1 only when **all** of the following hold, else 0:

- `this.sector === '2'` (else immediate 0);
- current AOM value is non-null and `>= 2.0` (floor);
- the unadjusted operating margin series is retrievable via `window.dataProcessor.getDataForAnalyzer('operating_margin');`
- **both** of 2024 and 2023 are present in that series (`recentOMYears.length >= 2`);
- **every** one of those two values is `<= -2.0` (ceiling).

Factor string: "OM-AOM divergence detected: AOM appears healthy but OM shows sustained losses". Header comment: "Fires when the ADJUSTED margin looks healthy (>= +2%) while the UNADJUSTED operating margin has been at or below -2% for two consecutive years - i.e. the adjustment, not the operations, is what makes the institution look solvent." The 2026-08-26 scoping comment is authoritative on why it is private-only:

> *"PRIVATE (FASB) ONLY. Measured on the current panel, 2026-08-26: sector 2 - unadjusted OM median +8.0%, 31% negative, 26% at or below -2%; sector 1 - unadjusted OM median -12.2%, 99% NEGATIVE, 76% at or below -2%. … Running this gate on publics fired it for 272 of 779 institutions (35%) on an accounting convention rather than on distress. It is scoped to FASB reporters … On the current panel it fires for 0 privates: correct and dormant, not dead. Re-check the counts above before widening scope."*

A separate 2026-08-26 comment records that the kicker "had never fired for any institution, ever, for two independent reasons" — `getOMDataForComparison` returned a raw row array that was then indexed by year string, and `measureAnalysis.institution` did not exist so the unitId lookup failed — and that both were repaired (the unitId is now stashed as `this.__unitId` in `analyze()`, line 194).

- **Maximum points**: `this.measureConfig.maxRiskPoints = 8`, for all sectors. The HTML wrapper re-asserts `maxRiskPoints = 8`. Framework: 8 in all four sectors.
- **Risk level mapping**: from **points as a ratio of `maxPoints` (8)**, all floors:

Risk level	Ratio floor	Points floor	Integer points
Critical	≥ 0.80	≥ 6.4	7, 8
High	≥ 0.55	≥ 4.4	5, 6
Moderate	≥ 0.30	≥ 2.4	3, 4
Low	—	< 2.4	0, 1, 2

`riskPercentage = Math.round((riskPoints / maxPoints) * 100)`.

- **Data-gap behaviour**: `createFallbackResult` returns `{ success: true, measure: 'operating_margin', institution: {...}, analysisResults: { riskPoints: 0, isDataUnavailable: true }, detailedReport: {...} }`. `isDataUnavailable` **is** set (`true`); **`riskLevel` is absent** from the object (no `'Unknown'`, no `'Informational'`). Same annotations as the parent file ("CHANGED: false → true", "CHANGED: 6 → 0"). Triggered when the institution row is not found or when `hasValidData` sees fewer than 2 non-null years. Separately, the HTML wrapper returns, when the dataset itself is missing or has fewer than 2 rows: `{ success: false, analysisResults: { riskPoints: 0, riskLevel: 'High' }, detailedReport: { headline: 'Adjusted operating margin data not available' } }` — note **`riskLevel: 'High'` on a data-absence path**, and no `isDataUnavailable` flag. `createErrorResult` is called (lines 198, 260) but **not defined** on this class.
- **Peer logic**: identical to Operating Margin. Peers are all rows whose `sector` cell equals `this.sector`, minus self, filtered by `carnegieBasic` array membership, `hbcuFilter` and `customPeerGroup`. No `StandardPeerBenchmarker`. The scored comparison (Factor 2) is a **percentile** banded into quartiles; peer **median** is used in report findings and peer **mean** in the `generatePeerComparison` string.
- **Notes**:
  - **`__rejectIsolatedBreaches` is applied here** (called at the end of `extractMeasureData`) and is not applied in the Operating Margin or Structural Margin analyzers. Its 2026-08-26 comment: "A margin cannot exceed 100% of revenue in either direction. After the row scale is applied, a year beyond +/-100% is either a genuine sustained deficit - in which case the neighbouring years are extreme too - or a single bad cell. Only the ISOLATED case is dropped: that year is set to null and the rolling average uses the years around it." Evidence recorded in the same comment: "adjusted operating margin 209 isolated breaches, 100 of them in FY2022 alone, against a flat baseline of ~4 a year. For 104 of those 106 institutions both 2021 and 2023

are in bound - single-cell errors, not the onset of distress." And: "BOTH are source-data defects. This is a guard so one bad cell cannot put an institution at -469% and drive its score; it is not a substitute for fixing the query that builds these files."

- **Known scaling defect, unresolved.** The 2026-08-26 `rowScale` block records: "Scripps College's adjusted operating margin row is decimal ratios in every year except 2022, which holds -14.487 — a PERCENTAGE (-14.5%) sitting in a decimal row, 43x the row's own median. Scaling the row by 100 turned that into -1448.7% and moved the institution's 3-year average from +8.5% to -469.6%." Prevalence given: "adjusted operating margin 67 rows, 78 cells (17 private C21)". The conclusion recorded is: "Nothing in the number itself distinguishes them. This is a source-data defect and must be fixed in the query that builds these files, not guessed at in the analyzer."
- **`measureName` collision.** This class sets `this.measureName = 'operating_margin'`, the same value the Operating Margin analyzer sets. The two are distinguished downstream by analyzer key (`adjustedoperatingmargin` vs `operatingmargin`), not by `measure`.
- `assessRollingAverageTrend` / `calculateRollingAverageTrendChange` use the same wrong `'2010-12'` / `'2021-23'` keys as the Operating Margin analyzer; see that section. Reporting only, not scoring.
- Per the 2026-08-26 NRR framework comment (quoted under Net Revenue Ratio), this analyzer's output feeds **eight** layer-2 gates: `checkFinancialDistress`, `checkSeverityCondition`, `checkA5`, `checkA8`, `checkA9`, `calculateP5`, `calculateP9`, `checkCriticalMassEscalation`.

Brand Pipeline - Acceptance & Yield

MARKET DEMAND & STUDENT SUCCESS

- **Class**: `UnifiedAcceptanceRateYieldAnalyzer` (wrapper: `UniversalAcceptanceYieldAnalyzer`, wrapper label `'Acceptance Rate & Yield Analysis'`, analyzer key `acceptanceyield`)
- **Data key**: The analyzer does not call `getDataForAnalyzer` itself. The wrapper (line 1883) calls `this.dataProcessor.getDataForAnalyzer('acceptance_yield')`, which maps to `/annualized.*apps.*admits.*enroll/i`.
- **What it measures**: "Brand strength" across three series read for 2014–2024 — application volume (`{year} app`), acceptance rate (`{year} accrate`) and yield rate (`{year} yield`) — scored on each series' **total change since 2014** relative to the peer mean change. Levels are reported but not scored.
- **Sector differences**: `riskFramework.maxRiskPoints = selectedSector === '1' ? 10 : 8` — public 10, every other sector 8. The wrapper constructor sets the same, and after the call overwrites `analysisResults.maxRiskPoints = this.sector === '1' ? 10 : 8`. The three factor budgets are derived from `maxPoints`, so they differ by sector (see table below). **Community colleges are excluded from this measure entirely** — `UniversalAcceptanceYieldAnalyzer` is in `CC_EXCLUDED_ANALYZERS` in `analysis_orchestrator.js` (*"Brand Pipeline - CCs are open access"*) and `risk_scoring_engine.js` declares 0 points for `communityCollege`. Declared framework points: public 10, private 8, for-profit 8, community college 0. The peer-relative thresholds (`critical: 15, high: 8, moderate: 3`) are the same for all sectors.
- **Scoring**: The three budgets are computed as

```
accPointsBudget   = Math.round(maxPoints * 0.3)
yieldPointsBudget = Math.round(maxPoints * 0.3)
appPoints         = maxPoints - accPointsBudget - yieldPointsBudget
```

	Public (max 10)	Private / for-profit (max 8)
`appPoints` (Factor 1)	4	4
`accPointsBudget` (Factor 2)	3	2
`yieldPointsBudget` (Factor 3)	3	2
Sum	**10**	**8**

Factor 1 — application growth vs peers. Runs only when `applications.totalChange !== null && peerAnalysis.success` and the peer mean is non-null. `peerDifference = ((institutionChange - peerMean) / Math.abs(peerMean || 1)) * 100`. Thresholds are **ceilings** (<=, negative side).

Condition	Public points	Private points
`peerDifference <= -15`	4 (`appPoints`)	4
`peerDifference <= -8`	3 (`round(appPoints*0.75)`)	3
`peerDifference <= -3`	2 (`round(appPoints*0.5)`)	2
`peerDifference < 0`	1 (`round(appPoints*0.25)`)	1
`peerDifference >= 0`	0	0

Factor 1 **fallback** — reached only via `else if (brandAnalysis.applications?.dataAvailable)`, i.e. when the peer analysis failed. Scored on the **raw** `appChange`. Thresholds are **ceilings** (<).

Condition	Public points	Private points
`appChange < -30`	4	4
`appChange < -10`	2 (`round(appPoints*0.5)`)	2
`appChange < 0`	1 (`round(appPoints*0.25)`)	1
`appChange >= 0`	0	0

Factor 2 — acceptance-rate change vs peers (rising acceptance rate = falling selectivity = risk). Thresholds are **floors** (>=).

Condition	Public points	Private points
`peerDifference >= 15`	3 (`accPoints`)	2
`peerDifference >= 8`	2 (`round(accPoints*0.67)`)	1
`peerDifference >= 3`	1 (`round(accPoints*0.33)`)	1
`peerDifference < 3`	0	0

Factor 3 — yield-rate change vs peers. Thresholds are **ceilings** (<=, negative side).

Condition	Public points	Private points
`peerDifference <= -15`	3 (`yieldPoints`)	2
`peerDifference <= -8`	2 (`round(yieldPoints*0.67)`)	1
`peerDifference <= -3`	1 (`round(yieldPoints*0.33)`)	1
`peerDifference > -3`	0	0

Factors 2 and 3 have **no absolute fallback**: if the peer analysis fails or the series is missing, they are skipped and award nothing.

The three components sum to exactly `maxPoints` in both sectors (10 and 8), so the declared maximum is reachable and `Math.min(riskPoints, maxPoints)` is a no-op.

- **Maximum points**: `riskFramework.maxRiskPoints` — 10 public, 8 otherwise. **Reachable in both** as of the 2026-08-29 change; the private budgets previously summed to 7. Reaching it requires all three peer-relative factors to be in their worst band simultaneously, which in turn requires a successful peer analysis (the Factor 1 fallback alone caps the measure at 4).
- **Risk level mapping**: **Two mappings exist and only the second is published.** `assessBrandRiskWithPeerContext` computes a proportional `riskLevel` (`>= maxPoints*0.75` High; `>= maxPoints*0.5` Moderate; `>= maxPoints*0.25` Low-Moderate; else Low) and returns it as `riskAssessment.riskLevel`. `analyze()` **ignores it** and publishes `this.determineRiskLevel(riskAssessment.totalRiskPoints)` instead, which is absolute and sector-blind:

Condition	riskLevel
`riskPoints >= 6`	`High`
`riskPoints >= 3`	`Moderate`
otherwise	`Low`

The `Low-Moderate` band therefore never reaches the published output. The `riskSummary` and `headline` strings in `detailedReport` also use `determineRiskLevel`, so the proportional level is unused everywhere except `riskAssessment.brandRisk`, which is not published either.

- **Data-gap behaviour**: `createErrorResult` (missing unitId, institution absent from the dataset, or `brandAnalysis.dataAvailable === false`) returns `riskPoints: 0`, `maxRiskPoints: this.riskFramework?.maxRiskPoints || 8`, `riskLevel: 'Unknown'`, `isDataUnavailable: true`. `dataAvailable` is `dataQuality > 0.3` where `dataQuality = (count of the three series with ≥ 2 usable years) / 3`, so **one of three series is enough to pass the gate** (0.333 > 0.3); the two missing series' factors are then skipped and award nothing. **No inner path awards points for missing data.** The wrapper's missing-institution early return publishes `riskPoints: 10, riskLevel: 'Critical'` with no `isDataUnavailable` — the full public maximum awarded for an absent institution object, and 10 against a private published maximum of 8.
- **Peer logic**: `findAndAnalyzePeerGroup` filters `this.institutions` (already sector-restricted by `extractAllInstitutions`) with an explicit self-exclusion (`inst.unitId !== institution.unitId`), then `peerCriteria.carnegieBasic` (array membership, string-compared), `peerCriteria.hbcuFilter` and `peerCriteria.customPeerGroup`. Each surviving peer's row is then re-analysed and only peers with `dataAvailable` are kept, so `peerCount` is the count of peers **with usable brand data**. The **scored** comparison in all three factors is against the peer arithmetic **mean** of the corresponding `totalChange`. `calculateStatistics` also computes a `median` and exposes `individualValues`, which `benchmarkValue` uses to derive quartile/percentile strings for the report — those strings do **not** feed the score. The institution **is** excluded from its own peer group.
- **2026-08 changes**: one block, 2026-08-29 (v49.1), at the head of `assessBrandRiskWithPeerContext` (lines 380–386) — **factor budgets now sum to the declared maximum**: *"the three factor budgets were each rounded independently,*

so for a maximum of 8 (private) they came to round(3.2)=3, round(2.4)=2 and round(2.4)=2 - a total of 7. The declared maximum of 8 was unreachable, and no private institution could ever score full marks on brand pipeline. Public (max 10) happened to work: 4 + 3 + 3 = 10. The remainder is now given to the applications factor, which carries the largest share, so the three budgets always sum to maxPoints exactly."

- **Notes**:
  - The Factor 1 fallback compares `appChange` — an **absolute count of applications** (`current - baseline`) — against thresholds of −30 and −10, which read as percentages and are labelled as such in the emitted strings (`'Critical decline in applications (>30%)'`, `'Significant decline in applications (10-30%)'`). Any institution losing more than 30 applications in absolute terms takes the full `appPoints` on this path. The peer path, which is the normal one, is not affected.
  - The Factor 1 fallback is unreachable when `applications.totalChange === null`, because `applications.dataAvailable` would then be false; in practice it fires only when the peer analysis fails.
  - `riskFramework.factors` (`applicationTrend: 3, acceptanceRateTrend: 3, yieldRateTrend: 2`) is declared but never read — the live budgets are the computed `appPoints` / `accPointsBudget` / `yieldPointsBudget`.
  - `analysisResults.currentApplications`, `currentAcceptanceRate` and `currentYieldRate` are assigned with `|| null`, so a genuine value of 0 publishes as `null`.
  - `determineOverallBrandRanking` and `calculatePeerRanking` are defined but never called.
  - `benchmarking.currentPerformance` is set to the **application** performance object, and `analysisResults.percentileRank` therefore reports the application-change percentile only, not a composite.
  - `findInstitutionData` falls back to column 0 with a console warning if no UNITID header is found.

Capitalization Ratio

LIQUIDITY & BALANCE SHEET

- **Class**: `UnifiedCapitalizationRatioAnalyzer`
- **Data key**: `'capitalization_ratio'` (wrapper `UniversalCapitalizationRatioAnalyzer`, HTML line 1209), plus `'unrestricted_net_assets'` and `'debt_revenue'` (fallback `'annualized_debt_and_d_r_consist_0_pv___pu'`) supplied as `adjustmentInputs`. Internal `this.measureName = 'capitalization_ratio'`. Wrapper label string is `'Capitalization Ratio Analysis'`.
- **What it measures**: Header comment: "Measures expendable net assets (F1A15 + F1A17) as a share of expendable net assets plus long-term debt — a leverage/debt-coverage ratio. Higher values = stronger financial position and debt coverage capacity." The report narrative describes it as "the share of expendable net assets and long-term debt that is covered by expendable equity (restricted-expendable + unrestricted net position)."
- **Sector differences**:
  - `sectorName`: `'1'` Public, `'2'` Private, `'3'` For-Profit, `'4'` Community College, else `'Unknown'` (this analyzer differs from the other four, which default the last case to Community College).
  - Factors 1 and 2 branch on `this.sector === '1'` **only**: Public uses `statusFromPercentile`; every other sector — including `'4'` — uses the absolute `assessCapitalizationStatus` bands. Comment: "Publics: GASB pension/OPEB allocation drives this ratio negative sector-wide (peer average ~ -5.9), so the absolute 0.25/0.40/0.60 bands collapse into a sign test. Score peer-relative position instead."
  - The restriction-adjusted capitalization ratio (Tier 3A.2) is computed only when `this.unaData && this.debtData && this.sector === '2'` (Private).
  - Factors 3, 4 and 5 are identical across all sectors.
- **Scoring**:

`maxPoints = measureConfig.maxRiskPoints = 10`. Points are fractional (0.5 steps).

Factor 1 — current value (2 points, "20% weight"). Runs if `currentValue !== null`.

Status	Points
critical	2
concerning	1.5
adequate	0.5
excellent	0 (factor text pushed, no points)
unknown	0, and no factor text

Factor 2 — 3-year rolling average (4 points, "40% weight"). Runs if `rollingAverage !== null`.

Status	Points
critical	4
concerning	3
adequate	1
excellent	0
unknown	0, and no factor text

Status for Factors 1 and 2 comes from one of two functions.

Public (sector `'1'`) — `statusFromPercentile`, on the corresponding `benchmarking.*Performance.percentile`:

Percentile band (ceiling)	Status
percentile < 10	critical
percentile < 25	concerning
percentile < 50	adequate
percentile >= 50	excellent
null / undefined / non-finite	unknown

All other sectors — `assessCapitalizationStatus`, using `measureConfig.riskThresholds` (`critical: 0.25, moderate: 0.40, adequate: 0.60, excellent: 0.60`), evaluated in this order:

Test (floor)	Status
ratio >= 0.60 (`excellent`)	excellent
ratio >= 0.60 (`adequate`)	adequate — **unreachable**, same value as the test above
ratio >= 0.40 (`moderate`)	concerning
otherwise	critical
null / undefined	unknown

So in practice, for non-public sectors: `>= 0.60` excellent, `0.40` to `< 0.60` concerning, `< 0.40` critical. The declared `critical: 0.25` threshold is never consulted, and the `'adequate'` status (0.5 pts on Factor 1, 1 pt on Factor 2) is unreachable outside the public percentile branch.

Factor 3 — trend direction (2 points, "20% weight"). All sectors:

`measureAnalysis.trend`	Points
`'declining'`	2
`'improving'`	0
anything else (`'stable'`, `'insufficient-data'`)	0.5

Trend is set from the full available span: `percentChange = ((newest - oldest) / Math.abs(oldest)) * 100`; `> 10` improving, `< -10` declining, else stable.

Factor 4 — volatility (1 point, "10% weight"). Population standard deviation of all year values:

Volatility (floor, exclusive)	Points
> 0.15	1
> 0.10 (and <= 0.15)	0.5
<= 0.10	0

Factor 5 — peer comparison modifier (1 point, "10% weight"). Requires `peerAnalysis.success` and a non-null `currentPerformance.percentile`:

Percentile (ceiling)	Points
< 25	1
< 50	0.5
>= 50	0

Sum is capped: `Math.min(riskPoints, 10)`.

Alternate ladder — `scoreAdjustedCapRatio` (Tier 3A.2, Private only). Substituted for the whole standard assessment when `adjustedRisk.riskPoints > riskAssessment.riskPoints` ("Swap in adjusted scoring when it is stricter than the standard"). Adjusted ratio = UNA ÷ (UNA + long-term debt).

A guard fires first: if `unrestrictedNA + longTermDebt <= 0`, the result is 10 points / Critical / 100% with `scoringBasis: 'restriction-adjusted (negative net position)'`. The comment reads: "when UNA is negative and |UNA| > debt, the denominator in una/(una + debt) goes negative and the ratio flips positive, scoring severe distress as Low. Not observed in current cohorts (0 of 99 adjusted records), but the construct permits it."

Otherwise:

Adjusted ratio (ceiling)	Points	riskLevel
< 0.20	10	Critical
< 0.35	8	High

Adjusted ratio (ceiling)	Points	riskLevel
< 0.50	6	High
< 0.65	3	Moderate
>= 0.65	0	Low

- **Maximum points**: `measureConfig.maxRiskPoints = 10`; returned as `maxPoints` from `assessMeasureRisk` and surfaced as `analysisResults.maxRiskPoints`. The adjusted scorer hard-codes `maxPoints: 10`. The HTML wrapper additionally forces `maxRiskPoints = 10` on success. Factor caps 2 + 4 + 2 + 1 + 1 sum exactly to 10.
- **Risk level mapping**: from `riskPercentage = (riskPoints / 10) * 100`, computed **before** the `Math.min` cap is applied to `riskPoints`:

Condition (floor)	riskLevel
riskPercentage >= 65	Critical
riskPercentage >= 45	High
riskPercentage >= 25	Moderate
otherwise	Low

On the adjusted path, `riskLevel` is assigned directly from the ratio band instead (table above); the 6-point band and the 8-point band both map to `'High'`.

- **Data-gap behaviour**: `createFallbackResult` (no data, UNITID column missing, institution not in dataset, "Insufficient capitalization ratio data") returns `success: false` and `analysisResults: { currentValue: null, riskPoints: 0, maxRiskPoints: 10, riskLevel: 'Unknown', isDataUnavailable: true }` plus an empty `peerGroup`. `createErrorResult` (missing unitId, thrown exception) returns the same `riskLevel: 'Unknown'` / `isDataUnavailable: true` shape without `currentValue`. The HTML wrapper's own missing-`unitid` return is `{ riskPoints: 0, maxRiskPoints: 10, riskLevel: 'Unknown' }` with **no** `isDataUnavailable` flag.
- **Peer logic**: `performPeerAnalysisWithNewFiltering` uses `StandardPeerBenchmarker` when that global is defined, otherwise falls back to `basicPeerFiltering` with a console warning. `StandardPeerBenchmarker.findAndAnalyzePeerGroup` (in `standard_peer_benchmarker.js`) excludes self, then filters on Carnegie basic, HBCU (`'hbcu-only'` / `'non-hbcu-only'`, skipped when the filter is `'all'`), `customPeerGroup`, and finally `inst.sector === this.sector`. The candidate pool itself (`extractAllInstitutions`) is already sector-filtered by `if (this.sector && sector !== this.sector) continue;`. The comparison used for scoring is a **percentile rank** computed inside this analyzer's own `benchmarkValue`: peers strictly below the institution ÷ peer count, rounded; `>= 75` Top quartile/Excellent, `>= 50` Above median/Good, `>= 25` Below median/Fair, else Bottom quartile/Concerning. Report text uses the peer **mean** ("vs peer average of ..."), while `generatePeerComparison` reports position against both mean and median.
- **Notes**:
  - Live diagnostic left in `computeRestrictionAdjustedCapRatio`: "DIAGNOSTIC: 68 institutions with negative UNA get no adjusted ratio. Debt years must match `/^20\d{2}\s*d$/` while UNA years match bare `/^20\d{2}$/`. If no debt year matches, this returns null silently and the institution falls back to the standard path. Remove once the cause is identified." It emits a `console.warn` per affected institution.
  - The adjusted-ratio year search walks back up to 5 years from the latest year column, requiring both a UNA value and a long-term debt value `> 0`; `yearFallback` triggers a note in the report.
  - `generateDetailedReport` prints the adjusted-capitalization arithmetic as `` `($${unaM}M unrestricted NA ÷ $${unaM}M + $${debtM}M long-term debt)` `` — `unaM` is interpolated for both the numerator and the first denominator term, which is correct for UNA/(UNA+debt), and `gapPct` is computed as `Math.round((gap / currentValue) * 100)` where `currentValue` may be negative for publics.
  - `findColumnIndex` in this file matches header names by **exact lowercase equality**, unlike the viability, primary-reserve and debt-revenue analyzers, which use `includes()`.
  - **2026-08-26** — `__unitRow` UNITID index added, same "pure performance" / FIRST-WINS comment as the viability analyzer.
  - `measureConfig.riskWeights` (0.2 / 0.4 / 0.2 / 0.1) is declared but never read; the weights are hard-coded as point values in `assessMeasureRisk`.

- Rolling 3-year averages are computed as strict consecutive triples over the *available* year list, so gaps in the panel are silently bridged (`years[i-2], years[i-1], years[i]` need not be consecutive calendar years).

Debt-to-Revenue Ratio

LEVERAGE & CAPITAL STRUCTURE

- **Class**: `SimpleDebtRevenueAnalyzer`
- **Data key**: `'debt_revenue'`, with fallback `'annualized_debt_and_d_r_consist_0_pv___pu'` (wrapper `UniversalDebtRevenueAnalyzer`, HTML lines 2035–2036). Internal `this.measureName = 'debt_revenue_analysis'`. Wrapper label string is `'Debt to Revenue Analysis'`.
- **What it measures**: Header comment: "Debt to revenue is a measure of an institution's ability to meet its debt obligations. Lower debt to revenue ratios are better." The analyzer scores **growth**, not level: "1. Shows debt growth for selected university, 2010-23 (total & percent increase)... 3. Shows debt to revenue growth for selected university, 2010-23". Source columns are `yyyy_d` (annual total debt) and `yyyy_dr` (annual debt-to-revenue ratio), scanned for years 2010–2024.
- **Sector differences**: The single largest sector split of the five measures. `this.maxRiskPoints = this.sector === '1' ? 8 : 10`, and the whole threshold config is chosen by the same test — sector `'1'` gets the "PUBLIC UNIVERSITY THRESHOLDS", **every other sector** (Private `'2'`, For-Profit `'3'` and Community College `'4'` alike) gets the "PRIVATE UNIVERSITY THRESHOLDS". There is no separate community-college configuration.

	Public (`'1'`)	All other sectors
Total max points	8	10
`debt_growth.maxPoints`	4	5
`debt_growth` good / moderate / poor / critical	20 / 40 / 80 / 999	15 / 30 / 60 / 999
`ratio_growth.maxPoints`	4	5
`ratio_growth` good / moderate / poor / critical	8 / 20 / 40 / 999	5 / 15 / 30 / 999

(An `excellent: 0` entry also exists in both threshold blocks but is never read; the `growthPercent <= 0` test short-circuits first. The `critical: 999` entry is likewise never read — it is the implicit `else`.)

- **Scoring**: Two independent components, summed.

Component 1 — `calculateDebtGrowthRisk(debtGrowthPercent)`. Percent change in total debt from the first to the last available year. Note this component compares `absGrowth = Math.abs(growthPercent)` against the thresholds after the sign test.

Condition	Public points (max 4)	Other-sector points (max 5)
growthPercent <= 0 (debt decreased)	0	0
absGrowth <= good (20 / 15)	`ceil(4 × 0.2) = 1`	`ceil(5 × 0.2) = 1`
absGrowth <= moderate (40 / 30)	`ceil(4 × 0.4) = 2`	`ceil(5 × 0.4) = 2`
absGrowth <= poor (80 / 60)	`ceil(4 × 0.7) = 3`	`ceil(5 × 0.7) = 4`
otherwise	4	5

Component 2 — `calculateRatioGrowthRisk(ratioGrowthPercent)`. Percent change in the debt-to-revenue ratio from the first to the last available year. Identical point structure, but it compares `growthPercent` directly (no `Math.abs`).

Condition	Public points (max 4)	Other-sector points (max 5)
growthPercent <= 0 (ratio improved)	0	0
growthPercent <= good (8 / 5)	1	1
growthPercent <= moderate (20 / 15)	2	2
growthPercent <= poor (40 / 30)	3	4
otherwise	4	5

All thresholds in both tables are **ceilings** (<=). A component is simply not computed (stays at its initialised 0) when its growth percentage is `null` — i.e. fewer than two years of that series, or a first-year value that is not `> 0`.

Total: `Math.min(debtGrowthRisk + ratioGrowthRisk, this.maxRiskPoints)`, with the inline comment "Enforce maximum cap to prevent impossible results like 12/5". The two component caps sum exactly to the total cap in both sector configurations (4+4=8, 5+5=10), so the cap only binds if a component is changed.

- **Maximum points**: `this.maxRiskPoints`, set in the constructor as `this.sector === '1' ? 8 : 10` with the inline comment "Public: 8, Private: 10". Unlike the other four measures, the HTML wrapper does **not** override this value.
- **Risk level mapping**: `determineRiskLevel(totalPoints)` computes `percentage = (totalPoints / this.maxRiskPoints) * 100` and applies **ceilings** — the opposite convention to the other four analyzers, which use floors:

Condition	riskLevel
percentage <= 25	Low
percentage <= 50	Moderate
percentage <= 75	High
otherwise	Critical

Concretely: Public (max 8) — 0–2 pts Low, 3–4 Moderate, 5–6 High, 7–8 Critical. Other sectors (max 10) — 0–2.5 Low, 3–5 Moderate, 6–7.5 High, 8–10 Critical.

- **Data-gap behaviour**: There is no separate fallback method; every data-gap path throws and is caught by the single `try/catch` in `analyze()`. Thrown conditions: "No debt revenue data available" (`this.hasData` false), "Institution ID column not found in debt data", and `Institution ${id} not found in debt revenue dataset`. The catch returns `success: false` with `analysisResults: { riskPoints: 0, maxRiskPoints: <8 or 10>, riskLevel: 'Unknown', isDataUnavailable: true, currentDebt: null, currentRatio: null, debtGrowthPercent: null, ratioGrowthPercent: null, peerComparison: null }`. The inline comments mark this deliberately: "No risk points for missing data", "Unknown, not Critical", "Flag for overall tool". Note: an institution that is *found* but has no usable year data does **not** throw — it returns `success: true` with 0 points and `riskLevel: 'Low'`, and no `isDataUnavailable` flag. Separately, the HTML wrapper returns `{ riskPoints: 12, riskLevel: 'Critical' }` — with no `maxRiskPoints` and no `isDataUnavailable` — when the institution object has no `unitid`.
- **Peer logic**: Two paths.
  - If `StandardPeerBenchmarker` is defined (`this.hasEnhancedPeerFiltering`), `this.institutions` is built by `extractAllInstitutions()` — which does **not** sector-filter — and the benchmarker is called as `findAndAnalyzePeerGroup({unitId: 'dummy'}, peerCriteria)`. Because the literal `'dummy'` is passed instead of the institution's UNITID, the benchmarker's self-exclusion step removes nobody, so the subject institution remains inside its own peer group. The benchmarker then applies Carnegie, HBCU (`'hbcu-only'` / `'non-hbcu-only'`), `customPeerGroup`, and `inst.sector === this.sector`.
  - Fallback path: raw data rows are filtered directly by sector column, Carnegie basic, and HBCU — where the HBCU test expects the codes `'1'` and `'2'`, not the `'hbcu-only'` / `'non-hbcu-only'` strings the other analyzers and `StandardPeerBenchmarker` use.
  - The comparison is a **mean**, not a median or percentile: `calculatePeerMetrics` computes `avgDebtGrowth`, `avgRatioGrowth`, `avgCurrentDebt`, `avgCurrentRatio` via `calculateAverage`, and `compareInstitutionToPeers` reports simple differences (institution − peer mean). Peer results are reported only, never scored.
- **Notes**:
  - `extractDebtData` discards any debt or ratio cell whose parsed value is `< 0` (`value >= 0` required), so negative reported values are treated as absent.
  - `data.sourcesFound` counts only the debt (`yyyy d`) columns, not the ratio (`yyyy dr`) columns, yet drives the reported `dataQuality`: `>= 10` Excellent, `>= 7` Good, `>= 4` Fair, else Poor.
  - Growth is measured from the earliest to the latest year present in each series independently, and only when the first value is `> 0`. There is no post-GASB trend window here, unlike the viability and primary reserve analyzers.
  - `calculatePeerMetrics` looks up each peer with `this.findInstitutionRow(peer.unitId || peer.UNITID)`. On the fallback peer path the elements of `peers` are raw data **rows** (arrays), which have no `unitId` or `UNITID` property, so the lookup receives `undefined` and returns `null`; no peer metrics are collected on that path.

- `findColumnIndex` matches by substring (`includes`), so the debt column search terms `` `${year} d` `` / `` `${year}d` `` can match a `yyyy dr` header if the `yyyy d` header is absent — the ratio column search uses the more specific `` `${year} dr` `` terms.
- A `console.log("🔍 DEBT BLUE DEBUG", ...)` diagnostic remains in `calculateRiskAssessment` and fires on every scored institution.
- `generateDetailedReport` builds `recommendations` as a permanently empty array; the comment reads "REMOVED: All recommendation generation logic eliminated / No recommendations.push() statements".
- The fallback-findings block references `riskAssessment.riskFactors`, but `calculateRiskAssessment` never sets a `riskFactors` property, so that branch never fires.
- The peer-comparison strings in the report are guarded with truthiness checks (`peerAnalysis.peerMetrics?.avgCurrentRatio ? ... : ''`), so a genuine peer average of exactly 0 suppresses the comparison text.
- No dated (2026-08) comments appear anywhere in this file, and it received neither the `__unitRow` UNITID index nor the `__memoRow` peer memo added elsewhere on 2026-08-26 — both `extractDebtData` and `findInstitutionRow` scan every row per lookup.

Endowment Dependency Ratio

REVENUE DEPENDENCE & PRICING

- **Class**: `UnifiedEndowmentDependencyAnalyzer` (wrapper: `UniversalEndowmentDependencyAnalyzer`)
- **Data key**: two calls in the wrapper (lines 1038–1039): `getDataForAnalyzer('endowment_payout')` → `payout`, and `getDataForAnalyzer('total_operating_revenue')` → `revenue`.
- **What it measures**: The share of total operating revenue funded by the endowment payout, computed within a single year as `Math.abs(payout[Y]) / total_operating_revenue[Y]` (IPEDS reports the payout as a negative outflow). The header comment frames ~15%+ as the point where the operating model has "structurally transitioned" from tuition-plus- endowment-supplement to endowment-supported.
- **Sector differences**:
  - `measureConfig.maxRiskPoints = (this.sector === '2') ? 6 : 0`.
  - `calculateRiskPoints()` returns immediately for any sector other than `'2'` with `riskPoints: 0, maxRiskPoints: 0, riskLevel: 'Informational', levelPoints: 0, levelLabel: 'N/A', informational: true`. **Confirmed: public and Community College genuinely score 0 and are never given points on any path.**
  - `analysisResults.isInformationalOnly` is set to `this.sector !== '2'`.
  - Sector row filtering: sector `'2'` pulls SECTOR rows `['2','5']`; sector `'1'` and sector `'4'` both pull `['1','4']`; any other sector pulls `[sector]`.
  - Header comment: input fields are F2H03C / F2D16 for private and F1H03C / F1B09 for public; data window ~4 years (2021-2024) private, ~3 years public. The code does not itself reference those IPEDS field names — it takes whatever year columns the two supplied datasets carry.

Scoring (private / sector '2' only)

Single component ("Level"), evaluated in declaration order with `break` on first match, so every band is a **floor** (`pct >= min`):

Condition	Direction	Points	Label
dependency ≥ 0.16 (16%)	floor	6	`'Critical'`
0.12 ≤ dependency < 0.16	floor	4	`'High'`
0.08 ≤ dependency < 0.12	floor	2	`'Moderate'`
dependency ≥ 0 (below 8%)	floor	0	`'Low'`

`totalPoints = Math.min(levelPoints, 6)`. There is no trend, drift or peer component in the score; `driftPP` and `trendDirection` are computed and reported but the comment at line 250 states they do not affect scoring in v1.

- **Maximum points**: 6 for sector `'2'`; **0** for every other sector (`measureConfig.maxRiskPoints`).
- **Risk level mapping** (private only):

Condition	riskLevel
totalPoints ≥ 6	`'Critical'`
4 ≤ totalPoints < 6	`'High'`
2 ≤ totalPoints < 4	`'Moderate'`
totalPoints < 2	`'Low'`

Non-private sectors return the literal string `'Informational'`.

- **Data-gap behaviour**: `createErrorResult()` returns `success: false`, `riskPoints: 0`, `maxRiskPoints: this.measureConfig.maxRiskPoints`, `riskLevel: 'Unknown'`, `isDataUnavailable: true`. Triggered by `MISSING_UNITID`, `NO_DATA` (UNITID column missing in either source; institution absent from either source; or no year in which both sources give a finite value with `revenue > 0`), and `ANALYSIS_ERROR`. The `'Informational'` return for non-private sectors does **not** set `isDataUnavailable`.
- **Peer logic**: `findPeerGroup()` ignores `peerCriteria` entirely. The group is every institution in the (sector-filtered) dataset with a different unitId, the same `sector` string, and the same `carnegie` string. Peer statistics are computed on each peer's current dependency percentage and returned as mean, median (`calculatePercentile(…,50)`),

25th and 75th percentile plus the sorted values. The **comparison printed in the report is against the median**; nothing peer-derived enters the score.

- **Notes**:
  - Header comment records the thresholds as "IPEDS-scaled (Option 1)": F2D16 includes total investment return, so 16% IPEDS-dependency ≈ 19% audit-derived dependency.
  - Declared Critical Mass domain: `revenue` (explicitly *not* `balance_sheet`).
  - No `2026-08-` dated change block appears in this file.

Endowment Draw Sustainability

LIQUIDITY & BALANCE SHEET

- **Class**: `UnifiedEndowmentDrawSustainabilityAnalyzer` (wrapper: `UniversalEndowmentDrawSustainabilityAnalyzer`)
- **Data key**: two calls in the wrapper (lines 1070–1071): `getDataForAnalyzer('endowment_payout')` → `payout`, and `getDataForAnalyzer('endowmentTotal')` → `endowmentTotal`.
- **What it measures**: The annual endowment draw rate, `Math.abs(payout[Y]) / endowmentTotal[denominatorYear]`, against the NACUBO-sustainable 4.5–5% band, plus whether that rate is accelerating.
- **Sector differences**:
  - `measureConfig.maxRiskPoints = (this.sector === '2') ? 10 : 0`.
  - `calculateRiskPoints()` returns immediately for any sector other than `'2'` with `riskPoints: 0, maxRiskPoints: 0, riskLevel: 'Informational', levelPoints: 0, levelLabel: 'N/A', trendPoints: 0, trendReason: 'N/A', informational: true`. **Confirmed: public and Community College genuinely score 0 on every path.**
  - **Denominator year differs by sector** and this applies to the reported rate even for the informational sectors: `denominatorYear = (this.sector === '2') ? year - 1 : year` (private uses prior-year EOY as a proxy for BOY; public/CC uses the same year, because F1H01 is already a BOY figure).
  - Sector row filtering: sector `'2'` → SECTOR rows `['2','5']`; sector `'1'` and sector `'4'` → `['1','4']`; else `[sector]`.
  - Note: the `isPrivate` test that selects the denominator is `this.sector === '2'`, so a sector-5 row analysed under a sector-`'1'`/`'4'` run would take the same-year denominator.

Scoring (private / sector '2' only)

Level component, evaluated on `referenceRate = Math.max(currentRate, threeYearAvg)`, in declaration order with `break` on first match — every band is a **floor**:

Condition	Direction	Points	Label
referenceRate ≥ 0.065 (6.5%)	floor	8	`'Critical'`
0.055 ≤ referenceRate < 0.065	floor	6	`'High'`
0.050 ≤ referenceRate < 0.055	floor	4	`'Moderate'`
0.045 ≤ referenceRate < 0.050	floor	2	`'Low-Moderate'`
referenceRate ≥ 0 (below 4.5%)	floor	0	`'Low'`

Trend component, mutually exclusive `if / else if` — both are strict floors (`> X`, not `>=`):

Condition	Direction	Points
yoyChange non-null and > 0.010 (1.0 pp)	strict floor	2
else `trendSlope` non-null and > 0.003 (0.3 pp/yr)	strict floor	1
otherwise	—	0

yoyChange = latest year's draw rate minus the previous available year's (needs ≥2 computable years). `trendSlope` = ordinary-least-squares slope of drawRate on year over all computable years (needs ≥2). `threeYearAvg` = mean of the last up-to-3 computable years — the comment states it uses "whatever's available if fewer than 3".

`totalPoints = Math.min(levelPoints + trendPoints, 10)`. Maximum attainable 8 + 2 = 10.

- **Maximum points**: 10 for sector `'2'`; **0** for every other sector.
- **Risk level mapping** (private only):

Condition	riskLevel
totalPoints ≥ 8	`'Critical'`
5 ≤ totalPoints < 8	`'High'`

Condition	riskLevel
3 ≤ totalPoints < 5	`'Moderate'`
totalPoints < 3	`'Low'`

Matches the header comment "0-2 Low, 3-4 Moderate, 5-7 High, 8-10 Critical". Non-private sectors return `'Informational'`.

- **Data-gap behaviour**: `createErrorResult()` returns `success: false`, `riskPoints: 0`, `maxRiskPoints: this.measureConfig.maxRiskPoints`, `riskLevel: 'Unknown'`, `isDataUnavailable: true`. Triggered by `MISSING_UNITID`, `NO_DATA` (UNITID column missing in either source; institution absent from either source; or no year survives — a year is dropped when the payout is NaN, the required denominator year is absent from `endowmentTotal`, or the denominator is NaN or ≤ 0), and `ANALYSIS_ERROR`. The non-private `'Informational'` return does not set `isDataUnavailable`.
- **Peer logic**: identical mechanism to Endowment Dependency — `findPeerGroup()` ignores `peerCriteria` and matches on same `sector` string + same `carnegie` string, excluding self. Statistics are mean, median, p25, p75 on peers' `currentRate`. Report text compares against the **median**. Nothing peer-derived enters the score.
- **Notes**:
  - Declared Critical Mass domain: `balance_sheet`.
  - Header comment: private F2H03C ~4 years (2021-2024), public F1H03C 2022-2024; "Limited historical depth means we lose long-term trend confidence, but the YoY jump signal compensates for catching acceleration events."
  - No `2026-08-` dated change block appears in this file.

Endowments per FTE

LIQUIDITY & BALANCE SHEET

- **Class**: `UnifiedEndowmentAnalyzer` (wrapper: `UniversalEndowmentAnalyzer`)
- **Data key**: The analyzer does not call `getDataForAnalyzer` itself. The wrapper (`modular_tool_v48 no data cap.html`, lines 1928–1949) calls it four times and selects with `||`:
  - `perFTE = getDataForAnalyzer('endowments_per_fte') || getDataForAnalyzer('endowmentPerFTE') || getDataForAnalyzer('endowments')`
  - `totalEndowment = getDataForAnalyzer('endowmentTotal') || getDataForAnalyzer('endowments')`

(The risk-adjustment canonical map in `analysis_orchestrator.js` separately labels this measure `'endowments_per_fte'`.)

- **What it measures**: Two things scored together — the current level of endowment per FTE (latest year column present) and the total-endowment percent change from the earliest to the latest year column. The file header states the inflation benchmark as "45% growth (2010-2024) maintains purchasing power"; the code applies 45 only to non-public sectors (see Scoring).
- **Sector differences**:
  - `isPrivate = (this.sector === '2')`. Maximum is 12 for private, 6 for every other sector the analyzer is instantiated for.
  - Per-FTE dollar cut-points differ: the public branch is selected by `this.sector === '1'`; **every other sector, including Community College (sector `'4'`), takes the `else` branch, which is the private dollar ladder** ($10K/$50K/$100K/$200K) combined with the public (6-point) tier values. See Notes.
  - Growth benchmark: `inflationBenchmark = this.sector === '1' ? 240 : 45`. Community College therefore uses 45, like private.
  - The 2026-08-28 comment block (CHANGE 1B) records that scoring publics on endowment level is deliberate; the former "Informational Only" labels and the note saying publics are not penalised were deleted.
  - The orchestrator instantiates this analyzer for sectors `'1'`, `'2'` and `'4'` only.

Scoring

Point ladders are declared explicitly (lines 405–417):

Component	Private (sector 2)	Public / other (sector 1, 4)
`basePerFTE`	6	3
`peerPerFTE`	2	1
`baseGrowth`	3	1
`peerGrowth`	1	1
Sum	**12**	**6**

Factor 1 — Per-FTE Endowment Level. Scored only when `perFTETrends.currentValue !== null`. All bands are ceilings (`value < X`), with the top band an open floor.

Public branch (`this.sector === '1'`), tiers `[3, 2, 2, 1, 0]`:

Condition	Direction	Points
perFTEValue < 4000	ceiling	3
4000 ≤ perFTEValue < 6000	ceiling	2
6000 ≤ perFTEValue < 10000	ceiling	2
10000 ≤ perFTEValue < 15000	ceiling	1
perFTEValue ≥ 15000	floor	0

Private / all other sectors branch, tiers `[6, 4, 3, 2, 0]` for private and `[3, 2, 2, 1, 0]` for sector 4:

Condition	Direction	Points (private)	Points (sector 4)
perFTEValue < 10000	ceiling	6	3
10000 ≤ perFTEValue < 50000	ceiling	4	2
50000 ≤ perFTEValue < 100000	ceiling	3	2
100000 ≤ perFTEValue < 200000	ceiling	2	1
perFTEValue ≥ 200000	floor	0	0

Peer adjustment added to Factor 1 (from `benchmarking.currentPerformance.quartile`):

Quartile string	Private	Public / sector 4
`'Bottom quartile'`	+2 (`peerPerFTE`)	+1
`'Below median'`	`+Math.floor(2 * 0.5)` = +1	`+Math.floor(1 * 0.5)` = **+0**
anything else	+0	+0

Factor 1 is then clamped: `Math.min(perFTEPoints, basePerFTE + peerPerFTE)` — 8 private, 4 public/sector 4.

Factor 2 — Total Endowment Growth. Scored only when `totalTrends.percentChange !== null`. `inflationBenchmark` = 240 for sector `'1'`, 45 otherwise. Bands are ceilings except the last.

Condition	Direction	Points (private, tiers `[3,2,1,1,0]`)	Points (public/sector 4, tiers `[1,1,1,0,0]`)
growth < 0	ceiling	3	1
0 ≤ growth < 0.5 × benchmark	ceiling	2	1
0.5 × benchmark ≤ growth < 0.75 × benchmark	ceiling	1	1
0.75 × benchmark ≤ growth < benchmark	ceiling	1	0
growth ≥ benchmark	floor	0	0

Benchmark values in force: public (`sector '1'`) 240% → cut-points 120% / 180% / 240%. Private and sector 4: 45% → cut-points 22.5% / 33.75% / 45%.

Peer adjustment added to Factor 2 (from `benchmarking.growthPerformance.quartile`):

Quartile string	Private	Public / sector 4
`'Bottom quartile'`	+1 (`peerGrowth`)	+1
`'Below median'`	`+Math.floor(1 * 0.5)` = **+0**	**+0**
anything else	+0	+0

Factor 2 is clamped: `Math.min(growthPoints, baseGrowth + peerGrowth)` — 4 private, 2 public/sector 4.

Terminal `totalPoints = Math.min(totalPoints, maxPoints)`; the in-code comment states this is now a defensive no-op because the ladders sum to exactly `maxPoints`.

- **Maximum points**: `maxPoints = isPrivate ? 12 : 6`, returned as `analysisResults.maxRiskPoints` and `detailedReport.maxRiskScore`.
- **Risk level mapping** (`determineRiskLevel(points)`, lines 941–946), on the raw point total, **not** normalised to the sector cap:

Condition	riskLevel
points ≤ 2	`'Low'`
3 ≤ points ≤ 6	`'Moderate'`
7 ≤ points ≤ 10	`'High'`
points ≥ 11	`'Critical'`

Because publics and Community Colleges cap at 6, they can never be labelled `'High'` or `'Critical'` on this measure.

- **Data-gap behaviour**: `createErrorResult()` returns `success: false`, `analysisResults.riskPoints: 0`, `maxRiskPoints: this.sector === '2' ? 12 : 6`, `riskLevel: 'Unknown'`, `isDataUnavailable: true`. Triggered by `MISSING_UNITID`, `NO_DATA_AVAILABLE` (validation `dataQuality === 'No Data'`), `INSTITUTION_NOT_FOUND`, `ANALYSIS_FAILED`. A *partial* gap (institution found, but one of the two metrics null) is not an error: the corresponding factor is skipped, contributes 0 points and pushes no breakdown row, and the result is a normal success with `riskLevel` derived from whatever scored (0 points → `'Low'`).
- **Peer logic**: `findAndAnalyzePeerGroup()` builds the group from `extractAllInstitutions()`, which retains only rows whose SECTOR column equals the selected sector, then excludes self and applies (only if supplied in `peerCriteria`) a Carnegie `c21basic` list filter, an HBCU filter (`'hbcu-only'` / `'non-hbcu-only'`), and a `customPeerGroup` unitId whitelist. Statistics computed on the group include mean, median, 25th and 75th percentile. The scoring path uses a **percentile**: `benchmarkValue()` computes `percentile = round(count(peers strictly below institution) / n × 100)`, higher = better for endowment, and maps ≥75 → `'Top quartile'`, ≥50 → `'Above median'`, ≥25 → `'Below median'`, else `'Bottom quartile'`. Only the two lowest quartile strings add points. Narrative text elsewhere quotes the peer **median**; the `generatePeerComparison()` text uses the peer **mean**.
- **Notes**:
    - Changes dated **2026-08-28 (v49.1)** in the code, headed CHANGE 1A and CHANGE 1B (lines 355–396). CHANGE 1A replaced fraction-of-max weights whose declared maxima summed to 15 (private) and 8 (public) against caps of 12 and 6, with explicit integer ladders; the comment records that 42 of 998 privates reached a raw 13–15 and 29 of 554 publics a raw 7–8 and were all clamped. CHANGE 1B removed the "Informational Only" assessment and the note claiming publics were not penalised, citing 458 of 554 baseline publics scoring above 0, 92 at 6/6 and 63 at 5/6.
    - The file header (lines 15–18) still says "Per FTE Endowment: 0-8 points" and "Total Growth: 0-5 points"; the code's Factor 2 maximum is 4 (private) / 2 (public).
    - Inline comments at lines 425 and 474 say "private max 8 of 12" and "private max 4 of 12" — these match the code.
    - `filterDataBySector()` matches the SECTOR column against the selected sector string exactly; it does not fold sector 5 into 2 or sector 4 into 1 (unlike the dependency / draw / restriction analyzers, which do).
    - `calculateEnhancedTrends()` computes `percentChange` only when `baselineValue && currentValue && baselineValue > 0`; a zero baseline yields null.
    - The wrapper's missing-institution guard returns `{ riskPoints: 13, riskLevel: 'Critical' }` — 13 exceeds the declared 12-point cap.

Enrollment Trends

MARKET DEMAND & STUDENT SUCCESS

- **Class**: `UnifiedEnrollmentAnalyzer` (real analyzer); `UniversalEnrollmentAnalyzer` (wrapper, `modular_tool_v48 no data cap.html` lines 1386–1442). Orchestrator key `enrollment`.
- **Data key**: `'enrollment'`. Called in the wrapper at line 1411: `this.dataProcessor.getDataForAnalyzer('enrollment')`. Resolved in `data_processing_pipeline.js` line 24 by `/annualized.*GD.*UG.*enrollment.*pv.*(and|&).*pu/i`.
- **What it measures**: the fact of enrolment decline over two windows — long term (2010→2024) and recent (5-year) — with the institution's position against its peer group applied as a modifier rather than a co-equal signal. The header states the rationale directly: *"An institution down 8% while its peer group is down 20% is still down 8%, and the peer group being worse does not pay the bills — so the absolute decline scores, and outperforming the peer group removes the modifier rather than earning a credit."* Enrolment is **FTE, not headcount** (*"median ratio 0.808 to the IPEDS headcount figure. This is deliberate; the tiers below are fitted to the observed FTE distribution across 1,491 institutions."*)

Sector differences

`getMaxRiskPoints()` (lines 307–315) returns 15 for sector `'1'`, 18 for `'2'`, 15 for `'4'`, and 15 for anything else. Ladder selection (line 201–202) is by `isPublicLike = (this.sector === '1' || this.sector === '4')`: public/CC use the `public` ladder, everything else uses the `private` ladder.

Sector `'3'` (for-profit) therefore takes the **private ladder (18 points of tiers) against a maximum of 15**, and is clipped by the `Math.min` at line 282.

The wrapper (lines 1417–1424) overrides the published `maxRiskPoints` to `sector '1' → 15, '2' → 18, '4' → 12, else 15` — see Defects.

Scoring

Ladders live in `this.riskConfig.ladders` (lines 36–52). Each ladder is `[threshold, points]` evaluated in order by `score()` (lines 220–223): `for (const t of tiers) { if (v <= t[0]) return t[1]; } return 0;`. **Every threshold is a ceiling: the tier fires when `value <= X`.** Values are percentages.

The tiers are stated to be fitted to the 2026 baseline distribution across 1,491 institutions (lines 27–32):

Series	p10	p25	p50	p75
long-term 2010–2024	−43.6%	−27.5%	−8.8%	+14.2%
short-term 5-year	−31.1%	−20.1%	−6.5%	+5.7%
peer gap (pp)	−26.5	−14.8	−2.2	+9.0

Factor 1 — long-term absolute decline, 2010→2024 (`totalChangePercent`):

Condition	public / CC	private
value <= −40	5	6
value <= −25	4	5
value <= −15	2	3
value <= −5	1	1
otherwise	0	0

Factor 2 — short-term absolute decline, 5-year window (`shortTerm.totalChangePercent`):

Condition	public / CC	private
value <= −25	4	5
value <= −15	3	4
value <= −8	2	2
value <= −3	1	1

Condition	public / CC	private
otherwise	0	0

Factor 3 — sustained decline across both windows (line 256). Fires when `Lv !== null && Sv !== null && Lv <= -3 && Sv <= -3`. Both are ceilings.

Condition	public / CC	private
long-term <= −3 **and** short-term <= −3	2	2

Factor 4 — peer-relative modifier, in PERCENTAGE POINTS (lines 261–280). `gapPP = Sv − peerShort`, where `peerShort` is the peer group's **mean** 5-year change. Negative means the institution did worse. *"Outperforming scores zero; there is no credit for beating peers."*

Condition	public / CC	private
gap <= −10 pp	4	5
gap <= −5 pp	3	3
gap <= −2 pp	1	1
otherwise	0	0

If `Sv` is present but no peer short-term mean exists, the factor is skipped and a "Peer comparison unavailable" note is pushed. **No points are awarded for the missing peer comparison.**

Do the components sum to the declared maximum? Yes. Public/CC: 5 + 4 + 2 + 4 = **15**. Private: 6 + 5 + 2 + 5 = **18**. The header states this was the point of the rebuild: *"The private allocation of 18 was previously unreachable: the four factors summed to at most 16… Both sector ladders now total exactly their framework maximum."*

Header-stated weighting: *"roughly 73/27 absolute to peer-relative"* — public 11/15 vs 4/15; private 13/18 vs 5/18.

Maximum points

Declared by `getMaxRiskPoints()`: **15 public, 18 private, 15 community college.** (`this.riskConfig.maxPoints` is a separate field hard-coded to 18; it is used only by `createErrorResult()` — see Defects.)

Reachable? Yes for public/CC and private. The maximum requires long-term <= −40, short-term <= −25, and peer gap <= −10 pp simultaneously; the sustained combo (<= −3 on both) is implied by the first two, so all four fire together. For sector `'3'` the private ladder can produce 18 against a maximum of 15, so 15/15 is reachable and the top three points are unattainable-by-being-clipped.

Risk level mapping

Lines 284–295. `riskPercent = (totalPoints / maxPoints) * 100`.

riskPercent	riskLevel
<= 25	`Low`
<= 48	`Moderate`
<= 68	`High`
> 68	`Critical`

In points: public/CC (max 15) — Low 0–3, Moderate 4–7, High 8–10, Critical 11–15. Private (max 18) — Low 0–4, Moderate 5–8, High 9–12, Critical 13–18.

Data-gap behaviour

Three distinct paths.

1. **No usable enrolment series at all** (`Lv === null && Sv === null`, lines 210–218). Returns `totalPoints: 0`, `riskLevel: 'Unknown'`, `dataUnavailable: true`, which is published at line 133 as `isDataUnavailable: true`. **No points awarded.** The 2026-08-28 header calls this out: *"A missing enrolment series previously scored `totalPoints += 3` with the message 'unable*

to assess performance risk' — a data gap charged as risk. It now returns riskLevel 'Unknown' with dataUnavailable, so the orchestrator drops the measure from BOTH numerator and denominator." 2. **Institution not found in the dataset** → `createErrorResult()` (lines 715–732): `success: false`, `riskPoints: 0`, `riskLevel: 'Unknown'`, `isDataUnavailable: true`, and `maxRiskPoints: this.riskConfig?.maxPoints || 8` — i.e. **18 regardless of sector**. The wrapper then overwrites `maxRiskPoints` with its own sector value (lines 1417–1424), so the 18 does not reach the report. 3. **Wrapper-level gaps** (`modular_tool_v48 no data cap.html` lines 1391–1410). Both award points for missing data and neither sets `isDataUnavailable`:

  - institution object missing an ID → `{ riskPoints: 0, riskLevel: 'High' }`
  - `enrollmentData` absent or shorter than 2 rows → `{ riskPoints: 50, riskLevel: 'High' }`

Both early-return **before** the `maxRiskPoints` override block, so neither carries a max. The orchestrator falls back to the framework value and clamps: 50 becomes the full sector maximum. See Defects.

Peer logic

`findAndAnalyzePeerGroup()` (lines 411–470), over `this.institutions`, which `extractAllInstitutions()` (lines 58–85) has already restricted to rows whose `sector` column matches `this.sector`.

- **Self-exclusion: yes**, explicitly (lines 421–423, `beforeSelfExclusion`).
- Filters applied, all optional: `peerCriteria.carnegieBasic` (array, string-compared against the row's `c21basic`), `peerCriteria.hbcuFilter` (`'hbcu-only'` keeps `hbcu === '1'`; `'non-hbcu-only'` drops `hbcu === '1'`; default when the column is absent is `'0'`), and `peerCriteria.customPeerGroup` (unitId whitelist).
- **Statistics** (`calculatePeerAverages`, lines 473–539):
  - `currentTotal` — **mean** of peer 2024 totals
  - `medianCurrentTotal` — **median**, *"Median, not mean: peer enrolment distributions are right-skewed in every segment, so a mean bar would sit above most of the group."* Published as `peerMedianTotal` with `peerMedianTotalCount`.
  - `totalChangePercent` — **mean** of long-term changes
  - `shortTermChangePercent` — **mean** of 5-year changes; **this is the value the peer scoring factor uses**
  - `groupBaselineTotal` / `groupEndpointTotal` — sums over peers reporting **both** endpoints, used only for market share
- **Market share** (`calculateMarketShareChange`, lines 548–566): institution's share of (itself + peer group) at 2010 vs 2024, returned as the percent change in that share. Returns `null` when `groupMemberCount < 5` (*"Fail closed: a share change built on a thin or lopsided group is noise."*) or when any of the four inputs is not a finite positive number.

2026-08 changes

2026-08-27 (line 114, `peerMedianTotal`) — *"PUBLISHED FOR THE RESILIENCE GATES. R10 and R12 previously hard-coded enrolment >= 10,000 as their scale test, which made resilience available to 97% of public doctoral institutions and 0% of baccalaureate colleges. They now read this field. Publishing it here — rather than having the adjustment layer recompute a peer median — keeps this analyzer the sole producer of the value, and avoids the recurring defect in this codebase of a gate reading a field its producer never published."*

2026-08-27 (line 474, `peerCurrentTotals`) — collected in `calculatePeerAverages` rather than recomputed downstream *"so there is exactly one definition of 'the enrolment of a typical peer'."*

2026-08-28 (line 27, scoring ladders) — ladders introduced, fitted to the observed FTE distribution across the 1,491 institutions of the 2026 baseline, with the stated design that *"Absolute decline carries the measure; peer position is a modifier."*

2026-08-28 (line 131, `isDataUnavailable`) — *"a missing enrolment series must leave the framework denominator, not score as risk. The orchestrator reads this flag."*

2026-08-28 (lines 153–197, the redesign block; "Specified by D. Greenstein") — the authoritative statement of intent. Four claims:

- *Weighting* moved from 50/50 to *"roughly 73/27 absolute to peer-relative. It was previously 50/50 (8 peer-relative points against 8 absolute)."*

- *What was wrong*: the two peer-relative factors built `peerDifference` from four branches, two of which inverted the sign — *"both declining: the code computed (instSeverity - peerSeverity)/peerSeverity, which is ALREADY NEGATIVE when the institution declines less, and then negated it — turning an outperformer into an underperformer"*, and *"both growing: (instChange - peerAverage)/peerAverage is POSITIVE when the institution grows faster, and positive is the 'worse than peers' side of the ladder."* The consequence in the 2026 baseline: *"a mode at exactly 8 points (5+3, both peer factors maxed with no absolute decline) holding 185 institutions, among them Harvard, Yale, Columbia, Johns Hopkins and Arizona State. 105 of the 153 institutions that declined LESS than their peers over the short window were charged the full 3/3 for 'critical short-term underperformance'. Marian University, up 64% over fourteen years, scored 16 of 18. Columbia College, down 66%, scored 0."*
- *The fix*: *"The peer leg is now a single comparison in PERCENTAGE POINTS (institution change minus peer change), not a relative percentage. That is what removes the need for the four branches the sign errors lived in, and it behaves correctly when either side is zero or negative."*
- *Also fixed*: the data-gap scoring (see above) and the unreachable private maximum.

2026-08-28 (line 367, short-term window) — *"the recent window is now FIVE years (2019-2024), not six (2018-2024). If 2019 is missing the analyzer falls back to 2018 and records which base year was actually used, so the report states the real span rather than a hard-coded one."*

Notes

- `dataQuality` is `'High'` at >= 10 non-null year values, `'Moderate'` at >= 5, else `'Poor'`. `assessConfidence()` combines it with `peerCount >= 15`.
- Year extraction uses `findColumnIndex()`, which does a case-insensitive **substring** match (`h.includes(name)`), so `'2010 total'` would also match a column named `'2010 total headcount'`. First match wins.
- `extractEnrollmentData` uses truthiness tests on year values (`data.total[2010] && data.total[2024]`, `if (data.total[2024])`, `data.total[2024] || data.total[2023]`). A genuine enrolment of 0 is therefore treated as missing. Whether this can arise depends on the source file; not determinable from this analyzer.

Estimated Impacts of Changing Federal Policy Landscape

- **Class**: `FederalPolicyAnalyzer` (real analyzer, `federal_policy_analyzer.js` — note it carries no `Unified` prefix); `UniversalFederalPolicyAnalyzer` (wrapper, `modular_tool_v48 no data cap.html` lines 2733–2810). Orchestrator key `federalpolicy`.
- **Data key**: three, tried in order by the wrapper at lines 2759–2761: `getDataForAnalyzer('federal_risk') || getDataForAnalyzer('bbb_risk_factors') || getDataForAnalyzer('federal_policy')`. All three resolve to the **same regex** in `data_processing_pipeline.js` lines 27–29, `/BBB.*risk.*factor.*pv.*(and|&).*pu/i`, so the fallback chain can only ever return the same table. Declared `dataRequirements: ['BBB risk factors 2024 pv & pu.xls']`.
- **What it measures**: **nothing is scored.** The file header is explicit — *"PURPOSE: Reports federal policy exposure data across 5 key measures. SCORING: Returns 0 risk points - purely informational analysis"* — and the code matches it exactly. See below.

What it actually computes — stated plainly

`FederalPolicyAnalyzer` reads five values from the institution's row of the BBB table, computes one trend, computes peer statistics for the same five values, assembles a narrative and five fixed recommendations, and assigns a data-quality label. **It contains no threshold, no comparison against a cut-point, and no arithmetic that produces a point.** The string `riskPoints` appears four times in the file (lines 159, 531, 552) and is the literal `0` on every occasion; `maxRiskPoints` is the literal `0`; `riskLevel` is the literal `'Informational'`. There is no `determineRiskLevel` method, no `riskFramework` beyond `measureConfig.maxRiskPoints: 0`, and no branch anywhere that could set a non-zero value.

The five reported measures (`measureConfig.measures`, lines 41–72):

Key	Reported name	Column searched (in order)	Unit
`pellGrants`	Pell Grant Recipients	`pgrnt_p`, `pgrant_p`, `pell_grant`, `PGRANT_P`	%
`federalLoans`	Federal Loan Recipients	`floan_p`, `federal_loan`, `FLOAN_P`	%
`nonResident`	Non-Resident Students	`pcte12nr`, `non_resident`, `PCTE12NR`	%
`graduateEnrollment`	Graduate Enrollment	any header matching `/(\d{4})%GD/i` with year 2010–2024; most recent year wins	%
`govGrants`	Government Grants & Contracts	`ggc`, `gov_grants`, `GGC`	$

`extractTrendData` builds a `yyyy%GD` series and, when more than one year is present, returns `{ startValue, endValue, totalChange, annualChange, percentChange, yearSpan, direction }` where `direction` is `'Increasing'` / `'Decreasing'` / `'Stable'` on the sign of `totalChange`. That trend is printed in a key finding and is not scored.

`assessDataQuality` (lines 503–515) is the only banding in the file, and it labels data completeness, not risk:

Condition	Label
`completeness >= 0.8` **and** at least one trend present	`High`
`completeness >= 0.6`	`Good`
`completeness >= 0.4`	`Moderate`
otherwise	`Limited`

where `completeness = (non-null of the five measures) / 5`.

Sector differences

None in behaviour beyond peer-pool membership. `this.sector` sets `sectorName` (`'1'` Public, `'2'` Private, `'3'` For-Profit, anything else Community College) and is the filter used twice: at extraction (line 96, `sector === this.sector`, where `sector` falls back to `this.sector` when the file has no sector column) and again in the peer filter (line 315). There are no sector-varying thresholds because there are no thresholds.

Framework points are **0 in all four sectors** (`risk_scoring_engine.js` lines 48, 92, 124, 184, each annotated `category: 'Federal Policy Risk'`; the community-college entry adds the comment *"Informational only"*).

Scoring

There is no scoring table to give. Every path returns `riskPoints: 0` against `maxRiskPoints: 0`.

Three independent mechanisms enforce this:

Layer	Mechanism
Real analyzer	`analysisResults` literals at lines 159–161, and in both error constructors at lines 531–533 and 552–554
Wrapper	Lines 2789–2793: `if (result.success && result.analysisResults) { riskPoints = 0; maxRiskPoints = 0; riskLevel = 'Informational'; }`
Orchestrator	`analysis_orchestrator.js` lines 229–246: `frameworkMax === 0` ⇒ `isExcluded` ⇒ `finalMax = 0`, `pts = 0`, `riskLevel = 'Informational'`, `scoringDisabled = true`

Maximum points

`0`, declared in `measureConfig.maxRiskPoints` (line 40), in `this.maxRiskPoints` on the wrapper (line 2736), in every returned `analysisResults`, and in the framework for all four sectors. **The maximum is 0 and is trivially "reached" on every run.** In the denominator loop the expression `Number(0) || fallbackMax || 0` evaluates 0 as falsy and falls through to the framework value, also 0, so the measure adds nothing to `adjustedMaxPoints` whether it is treated as available or excluded.

Risk level mapping

There is no derivation. `riskLevel` is the string literal `'Informational'` on all five return paths in the analyzer and both in the wrapper; the orchestrator rewrites it to `'Informational'` again for framework-excluded measures. `getRiskLevelColor()` (`modular_tool_v48 no data cap.html` lines 3363–3371) has no entry for `'Informational'`, so the report renders it in the default grey `#6c757d`.

Data-gap behaviour

Path	`success`	`riskPoints`	`riskLevel`	`isDataUnavailable`
Normal completion	`true`	0	`'Informational'`	*not set*
`createNoDataResult` — institution not found in the BBB table	`true`	0	`'Informational'`	**`true`**
`createErrorResult` — throw inside `analyze()`	`false`	0	`'Informational'`	**`true`**
Wrapper, `!institution.unitid`	`true`	0	`'Informational'`	*not set*
Wrapper, no BBB data at all → `this.createNoDataResult(...)`	—	—	—	—
Wrapper `catch`	`true` (explicitly commented *"Return success but with 0 points"*)	0	`'Informational'`	**`true`**

No path awards points for missing data — no path awards points at all.

One live defect on the no-BBB-data path: the wrapper calls `this.createNoDataResult(institution.name)` at line 2765, but `createNoDataResult` is a method of `FederalPolicyAnalyzer`, not of `UniversalFederalPolicyAnalyzer` or `SectorAwareAnalyzer`. The call throws `TypeError: this.createNoDataResult is not a function`, which is caught by the wrapper's own `try` at line 2757 and converted into the `catch` row above. The observable result is the same shape, reached by an unintended route, with the error message *"this.createNoDataResult is not a function"* printed in `keyFindings`.

Peer logic

`performPeerBenchmarking()` (lines 303–384). `StandardPeerBenchmarker` is **not** used; this analyzer has its own filter.

The candidate pool is `this.institutions`, built by `extractAllInstitutions()` from the BBB file itself and already restricted at build time to `sector === this.sector` (line 96) — where `sector` is the row's own sector value if a sector column is found, else `this.sector` for every row, so a BBB file without a sector column yields a pool of every institution in it. Filters then applied, in order:

1. **Self-exclusion** (line 314): `inst.unitId === (institution.unitid || institution.unitId)`. Here `inst.unitId` is `row[unitIdIdx]?.toString()` (line 93 — a **String**) and `institution.unitid` is also a String, so **this comparison does match and the institution is genuinely excluded from its own peer group.** This is the only one of the six measures where self-exclusion is unambiguously effective. 2. `inst.sector !== this.sector` (redundant with the extraction filter). 3. Carnegie: `peerCriteria.carnegieBasic` (array) if present, else legacy `peerCriteria.carnegie`. 4. HBCU: `peerCriteria.hbcuFilter` (`'hbcu-only'` / `'non-hbcu-only'`, against `inst.hbcu === '1'`), else legacy `peerCriteria.hbcu`. 5. `peerCriteria.customPeerGroup`. 6. **Truncation** (lines 362–364): `if (peerCriteria.count && peers.length > peerCriteria.count) peers = peers.slice(0, peerCriteria.count)` — the first N in **file order**, not the N most similar.

`calculatePeerStatistics()` (lines 387–417) re-derives all five measures for every surviving peer and returns, per measure, `{ count, mean, median, min, max }`:

- **mean** — arithmetic mean, the only statistic used in the report text.
- **median** — `values[Math.floor(values.length / 2)]`, the **upper** middle value for even counts, not the average of the two middle values. It is computed and returned but never printed.

Peer figures appear only in `generateDetailedReport`, appended as *"(peer average: X)"* to each of the five findings.

2026-08 changes

None. This file contains no dated change block of any kind. Its filesystem timestamp is 2026-08-30 10:41 with no accompanying comment. The one relevant 2026-08 change lives elsewhere: `analysis_orchestrator.js` lines 234–242, dated `2026-08-28 (v49.1)`, which introduced the rule that rewrites a framework-excluded measure's `riskLevel` to `'Informational'`. Its stated reasoning concerns Net Revenue Ratio rather than Federal Policy — *"a framework-excluded measure must not keep a severity label … the zeroing works — all 1,552 institutions show 0.0/0.0 — but the accompanying riskLevel still read 'Critical' for 441 of them"* — but Federal Policy passes through the same branch, which is what sets its `scoringDisabled: true` flag.

Notes

- **Confirmed as requested: Federal Policy computes no risk score whatsoever.** It is a five-value data readout with peer means, one graduate-enrolment trend, a fixed narrative and five fixed recommendations that are identical for every institution (lines 485–491). The 0 in the framework is not a suppression of a working score; there is no score to suppress.
- `measureConfig.measures.pellGrants.column` declares `'pgrant_p'`, but `extractFederalPolicyData` searches `'pgrnt_p'` **first** (line 197). `measureConfig` is documentation only — nothing reads its `column` fields.
- `findColumnIndex` matches by lower-cased **substring**, so the three-letter probe `'ggc'` will bind to the first header containing those letters in that order as a contiguous run, and `'sector'`/`'Control'` likewise.
- The peer median is an upper median for even counts, and is never displayed.
- Peer truncation by `peerCriteria.count` takes the first N rows in file order.
- `calculatePeerStatistics` calls `findInstitutionRow` — a full linear scan of the BBB table — once per peer per measure, five times over. This analyzer was not included in the 2026-08-26/27 indexing pass applied to the programme analyzers.
- If `peerCriteria` is undefined, line 318 dereferences it and throws; the throw is caught inside `performPeerBenchmarking` and returns `{ count: 0, institutions: [], error }` with no `summary` key, so `analysisResults.peerComparison` becomes `undefined`.
- `extractAllInstitutions` defaults `carnegieBasic` to `'21'` and `hbcu` to `'2'` when those columns are absent, so a Carnegie filter on any code other than 21 would empty the peer group and an `'hbcu-only'` filter always would.

Institutional Aid

REVENUE DEPENDENCE & PRICING

- **Class**: `SimpleGrantFundingAnalyzer` (wrapper: `UniversalGrantFundingAnalyzer`, wrapper label `'Grant Funding Distribution Analysis'`, analyzer key `grantfunding`)
- **Data key**: The analyzer does not call `getDataForAnalyzer` itself. The wrapper (line 2272) calls `this.dataProcessor.getDataForAnalyzer('grant_distribution')`, which maps to `/annualized.*grant.*distribution.*sfa.*pv.*(and|&).*pu/i`.
- **What it measures**: `SFA{yearPair}_P1_IGRNT_P` — the percent of full-time first-time undergraduates **awarded** institutional grant aid, plus its annual trend. The file header is explicit and emphatic that this is a **headcount coverage rate, not a discount rate**, and that the export label "Institutional Aid" disagrees with the quantity: *"An institution giving every freshman a token $500 grant scores 100% here. The thresholds above are calibrated to coverage, not to discount rate. Do not 'fix' the label by changing the variable, and do not read these scores as discount-rate risk."* `SFA{yearPair}_P1_IGRNT_A` (average grant amount) is also extracted but **is not scored** — it appears in the narrative only.
- **Sector differences**: `this.riskConfig.maxPoints = selectedSector === '2' ? 2 : 6`. Sector '2' (private) gets its own award column throughout — the public awards scaled by 0.25 — so the private ladder sums to exactly 2.00. Public awards are unchanged and can total 8 raw against a 6-point cap. The wrapper overwrites `analysisResults.maxRiskPoints` with `this.sector === '1' ? 6 : 2`, so for-profit ('3') and community college ('4') publish a max of **2** while the analyzer itself computes and caps against **6** — the two disagree for sectors 3 and 4. `risk_scoring_engine.js` declares public 6, private 2, for-profit 2, community college **0** (`// Limited CC relevance`).
- **Scoring**:

Factor 1 (peer path) — peer-relative grant coverage. `peerDifference = ((currentPercentage - peerAverage) / peerAverage) * 100`. Bands are **floors** (>=), evaluated in order, first match wins.

Condition	Public points	Private (sector '2') points
`peerDifference >= 40`	5	1.25
`peerDifference >= 25`	3	0.75
`peerDifference >= 15`	2	0.50
`peerDifference >= -10`	1	0.25
otherwise	0	0

Factor 1 (fallback) — absolute coverage, used **only** when no usable peer average exists (`peerAverage` null/undefined/non-finite/`<= 0`). Bands are **strict floors** (>), first match wins; if none match, a floor award is still made.

Condition	Public points	Private points
`percentage > 85`	4	1.00
`percentage > 75`	3	0.75
`percentage > 65`	2	0.50
otherwise (`F1_ABSOLUTE_FLOOR`)	1	0.25

Factor 2 — annual change in coverage, percentage points per year (`grantAnalysis.percentageTrend`). Bands are **strict floors** (>).

Condition	Public points	Private points
`annualChange > 5`	3	0.75
`annualChange > 2`	2	0.50
`annualChange > -2`	1	0.25
otherwise	0	0
trend null/non-finite	skipped, 0	skipped, 0

Do the components sum to the declared maximum? Private: yes exactly — 1.25 + 0.75 = 2.00, and the trailing `Math.min` is a no-op there. Public: **no** — the peer path can total 5 + 3 = 8 and the fallback 4 + 3 = 7, both against a 6-point maximum, and `Math.min(totalPoints, 6)` clips the excess. The change comment states this is deliberate: *"still clipped to 6 — public behaviour deliberately left as-is per the change request."*

- **Maximum points**: `this.riskConfig.maxPoints` — 2 for sector '2', 6 otherwise. **Reachable in both.** Public reaches 6 through the cap (any peer-path combination totalling ≥ 6, e.g. 5 + 3 or 5 + 2 or 3 + 3). Private reaches exactly 2.00 only via the peer path's worst combination (1.25 + 0.75); on the **absolute fallback** the private maximum is 1.00 + 0.75 = 1.75, so `Critical` is unreachable without a peer average.
- **Risk level mapping**: proportional to `maxPoints`, computed on the capped total:

Condition	riskLevel
`cappedRiskPoints <= maxPoints * 0.33`	`Low`
`cappedRiskPoints <= maxPoints * 0.67`	`Moderate`
`cappedRiskPoints < maxPoints`	`High`
otherwise (`=== maxPoints`)	`Critical`

Public cut-points (max 6): ≤ 1.98 Low; ≤ 4.02 Moderate; 5 High; 6 Critical. Private cut-points (max 2): ≤ 0.66 Low; ≤ 1.34 Moderate; 1.50/1.75 High; 2.00 Critical.

- **Data-gap behaviour**: If `currentPercentage` is null/undefined/non-finite, `assessGrantRisk` returns immediately with `totalPoints: 0`, `riskLevel: 'Unknown'`, `isDataUnavailable: true`, and the riskFactor string `'Grant coverage data unavailable - measure not scored'`. `analyze()` copies that through as `isDataUnavailable: totalRisk.isDataUnavailable === true`. If no rows at all, `createErrorResult` gives `riskPoints: 0`, `maxRiskPoints: this.riskConfig.maxPoints`, `riskLevel: 'Unknown'`, `isDataUnavailable: true`. A missing **trend** skips Factor 2 and awards nothing. **No inner path awards points for missing data.** The wrapper's missing-institution early return does: `riskPoints: 5, riskLevel: 'High'`, no `isDataUnavailable`.
- **Peer logic**: `findAndAnalyzePeerGroup` filters `this.institutions` (already sector-restricted by `extractAllInstitutions`) with an explicit self-exclusion (`inst.unitId === institution.unitid → false`), a redundant `inst.sector !== this.sector` check, `peerCriteria.carnegieBasic` (array membership, string-compared), `peerCriteria.hbcuFilter`, and `peerCriteria.customPeerGroup`. The comparison statistic is an arithmetic **mean** (`peerPercentageSum / peerPercentageCount`), not a median or percentile. The institution **is** excluded from its own peer group.
- **2026-08 changes**: three blocks. 1. File header, 2026-08-28 (v49.1) — **name/quantity mismatch**: *"the export labels this measure 'Institutional Aid', which reads as aid as a share of tuition revenue. The scored quantity is SFA{yearPair}_P1_IGRNT_P: the percent of full-time first-time undergraduates AWARDED institutional grant aid — a headcount coverage rate, not a discount rate. The label and the quantity disagree; the quantity is correct and intentional."* 2. `assessGrantRisk` docblock, 2026-08-28 (v49.1) — **scoring fix** for the private scale. What was wrong: *"Both sectors shared one raw ladder: Factor 1 could award 5 and Factor 2 could award 3, i.e. 8 raw points, against `maxPoints: sector==='2' ? 2 : 6`. A trailing Math.min() then clipped the total. In the private sector that meant the MILDEST possible combination of signals — Factor 1 'near peer levels' (1) plus Factor 2 'modest growth' (1) — already reached 2 of 2 and published as 'Critical'. The measure carried no information."* Evidence cited: *"779 of 998 private institutions (78%) scored 2.0/2.0 'Critical'; 132 scored 1.0 and 87 scored 0.0 — three distinct values across the whole sector."* The same block states the data-gap rule: *"DATA GAPS NEVER SCORE (both sectors). Previously a missing coverage figure added 3 points and a missing trend added 1 — risk awarded for absent data."* 3. `analyze()` result assembly, 2026-08-29 (v49.1): *"calculateTotalRisk sets isDataUnavailable on its no-coverage path, but it was never copied onto analysisResults, which is the only place the orchestrator looks. A riskLevel of 'Unknown' on its own does NOT remove a measure from the denominator - only isDataUnavailable or not_applicable does - so an institution with no grant coverage figure kept the measure's points in its denominator and scored zero against them."*
- **Notes**:
  - `data.currentPercentage = data.percentageData[2023] || data.percentageData[2022] || null` — a genuine coverage of **0%** in 2023 is falsy and silently falls through to the 2022 value, or to `null` (which then routes to the data-gap path). Same pattern on `currentAmount`.
  - Peer averaging uses `if (peerData.percentageData[2023])`, so a peer with a genuine 0% is excluded from the mean.

- The institution's current value may come from **2023 or 2022**, but the peer mean is built **from 2023 only**. A 2022-sourced institution value is compared against a 2023 peer mean.
- The 2026-08-29 comment refers to a method named `calculateTotalRisk`; the method in this file is `assessGrantRisk`. There is no `calculateTotalRisk` in this file.
- `extractGrantData` sets `hasData = sourcesFound > 0` where `sourcesFound` counts percentage **and** amount columns, so an institution with only amount data passes the `hasData` gate and then correctly lands on the `!hasCoverage` data-gap return.
- For sectors '3' and '4' the wrapper publishes `maxRiskPoints: 2` while the analyzer caps at 6; a for-profit or CC institution can therefore publish a riskPoints value greater than its published maximum. (Sector '4' is declared at 0 points in `risk_scoring_engine.js`, and the analyzer is still instantiated for CCs — it is not in `CC_EXCLUDED_ANALYZERS`.)

Institutional Support

OPERATING PERFORMANCE & COST STRUCTURE

- **Class**: `UnifiedInstitutionalExpenditureAnalyzer` (real analyzer, `unified_institutional_expenditure_analyzer.js`); `UniversalInstitutionalExpenditureAnalyzer` (wrapper, `modular_tool_v48 no data cap.html` lines 1582–1616). Orchestrator key `institutionalexpenditure`.
- **Data key**: `'institutional_expenditure'`. The real analyzer does not call `getDataForAnalyzer`; the wrapper does, at line 1599: `workingDataProcessor.getDataForAnalyzer('institutional_expenditure')` where `workingDataProcessor = this.dataProcessor || window.dataProcessor`. Resolved in `data_processing_pipeline.js` line 17 by `/annualized.*pct.*institutional.*expenditure/i`. Declared `dataRequirements: ['annualized pct institutional expenditure.xls']`.
- **What it measures**: institutional support (administrative overhead) as a percentage of total expenditures, at the latest available year, plus the change in that percentage since the earliest available year. The header states the direction: *"Higher institutional expenditure = BAD (more overhead/inefficiency) … Increasing trend = BAD (growing bureaucracy)"*. Data format is documented as *"Whole numbers (15 = 15%, not 0.15)"*; `analysisYears` is 2010–2023 and `currentYear` is 2023.

Sector differences

One ternary, at line 47: (`this.sector === '1' || this.sector === '4'`) selects the public threshold set; everything else (sectors `'2'`, `'3'`, and any unrecognised value) takes the private set. Community colleges therefore use the public set. There is no separate for-profit set.

A second sector split at line 427 sets `optimalRange` — `[8, 15]` for public/CC, `[6, 12]` otherwise — used only to choose recommendation text, never to score.

Scoring

Two components. `currentRate` is the last non-null year in the series; `totalChange` is `current - baseline` in **percentage points**, where baseline is the first non-null year.

Component 1 — current level (lines 314–337). Evaluated only if `currentRate !== null`. Branches in order, first match wins. All are **floors**:

#	Threshold key	Public / CC (`sector '1','4'`)	Private and all others	Test	Points
1	`dangerous`	30	24	`currentRate >= threshold`	**4**
2	`critical`	25	20	`currentRate >= threshold`	**3**
3	`concerning`	20	16	`currentRate >= threshold`	**2**
4	`good`	15	12	`currentRate >= threshold`	**1**
5	—	—	—	otherwise	**0**

`excellent` (10 public / 8 private) is defined in `riskFramework` and **never read by any branch**. The in-code comments beside the threshold definitions (lines 48–60) state a different point ladder from the one the code implements — they say `0 / 1 / 3 / 5 / 6`, the code awards `0 / 1 / 2 / 3 / 4`.

Component 2 — trend (lines 339–359). Evaluated only if `totalChange !== null`. Branches in order:

#	Test	Floor / ceiling	Points
1	`totalChange >= trendThresholds.concerning` (**+5**)	**floor**	**2**
2	`Math.abs(totalChange) <= trendThresholds.stable` (**2**)	band, inclusive both ends	**1**
3	`totalChange <= trendThresholds.improving` (**–2**)	**ceiling**	**0**
4	otherwise (i.e. `2 < totalChange < 5`)	—	**1**

Branch 2 is tested before branch 3, so a change of exactly **–2 falls into branch 2 and scores 1, not 0**. Only a decline strictly greater than 2 points reaches branch 3. Branch 2 also means a **flat series scores 1 penalty point**: "Stable trend" is not free. Branch 4 is reachable, covering increases between 2 and 5 points.

`totalPoints = component1 + component2`, then `cappedRiskPoints = Math.min(totalPoints, 6)`.

Do the components sum to the declared maximum? Yes: 4 + 2 = 6, so the `Math.min` cap never binds.

Maximum points

Declared `this.riskFramework.maxRiskPoints = 6` (line 46), published in `analysisResults.maxRiskPoints` and re-asserted by the wrapper (line 1612, on the `result.success` path only). Framework declares 6 in all four sectors. The wrapper does **not** set `this.maxRiskPoints`, so the `SectorAwareAnalyzer` catch path would publish 15.

Reachable. 6 requires `currentRate >= 30` (public/CC) or `>= 24` (private) **and** `totalChange >= +5`. Nothing in the code prevents both holding at once.

Risk level mapping

`determineRiskLevel(totalRiskPoints)` (lines 564–568) — absolute points, not a percentage:

Condition	Level
`totalRiskPoints >= 5`	`High`
`totalRiskPoints >= 3`	`Moderate`
otherwise	`Low`

Cut-points: 0–2 Low, 3–4 Moderate, 5–6 High. **There is no `Critical` level**; this measure can never publish one. A parallel field `riskAssessment.expenditureRisk` uses the same 5/3 cuts and the same three labels.

Data-gap behaviour

Path	`success`	`riskPoints`	`riskLevel`	`isDataUnavailable`	In denominator?
`catch` in `analyze()` — including the thrown *"Institution data not found for UNITID"* when `findInstitutionData` returns null	`false`	0	`'Analysis Failed'`	**`true`**	No — excluded
Wrapper early return, `!institution.unitid`	`false`	3	`'Moderate'`	*not set*; no `maxRiskPoints` either	No — excluded on `!success`
Institution row found but fewer than 2 non-null years	`true`	0	`'Low'`	*not set*	**Yes — 0/6**

That third row is the live gap. `analyzeTimeSeries` returns `current: null, totalChange: null, dataAvailable: false` when `entries.length < 2`; both scoring components are then skipped by their `!== null` guards, `totalPoints` stays 0, and the measure publishes a successful 0 of 6 labelled `Low`. `dataAvailable: false` is present inside `analysisResults.expenditureAnalysis.expenditureRate` but nothing in the orchestrator reads it.

No path awards points for missing data.

Peer logic

The peer group is computed and reported but does not affect the score. `assessExpenditureRisk(expenditureAnalysis, peerAnalysis)` takes `peerAnalysis` as a parameter and never reads it — unlike its instructional sibling, which applies a peer-median reduction.

`performPeerAnalysisWithNewFiltering()` (lines 214–250) constructs `new StandardPeerBenchmarker(this.institutions, this.sector, this.sectorName)` directly — it does not use the `findAndAnalyzePeerGroup` method that `standardizePeerFiltering(this)` installed in the constructor. Filter chain inside the benchmarker: self-exclusion, `peerCriteria.carnegieBasic`, `peerCriteria.hbcuFilter`, `peerCriteria.customPeerGroup`, then `inst.sector === this.sector`. Peers are then re-read from the expenditure file and kept only if `expenditureRate.dataAvailable`.

If the benchmarker returns an empty set, control **falls through to `performBasicPeerAnalysis`** (lines 252–278), which applies **only** the sector filter and a self-exclusion — no Carnegie, no HBCU, no custom peer group. A Carnegie filter that matches nobody therefore silently widens the peer group to the entire sector rather than reporting no peers.

Is the institution excluded from its own peer group? No. `extractAllInstitutions()` line 145 sets `unitId: this.parseNumericValue(row[unitidIndex])` — a **Number**. The subject object passed in is `{ unitId: unitid, unitid: unitid }` (line 218) where `unitid` is `institution.unitid`, a **String** (pipeline line 241). The benchmarker's self-exclusion is `inst.unitId !== institution.unitId && inst.unitId !== institution.unitid` — Number `!==` String is always true, so **no institution is ever removed from its own peer group**. The same mismatch defeats `performBasicPeerAnalysis`'s `inst.unitId !== institution.unitid`.

Statistics. `analyzePeerMeasureData` → `calculateStatistics` (lines 541–556) returns `{ count, mean, values }` for `currentRate` and for `totalChange`. **No median is computed** in this analyzer. `mean` is arithmetic. `performExpenditureBenchmarking` additionally produces a rank string via `calculatePeerRanking(..., false)` (lower is better), formatted `${rank}/${total} (${percentile}th percentile)` where `percentile = Math.round((betterThanCount / peerValues.length) * 100)` and `total = peerValues.length + 1` — so rank and percentile use different denominators. Benchmarking is suppressed entirely when fewer than 3 valid peers.

`analysisResults.currentRate` and `analysisResults.peerCurrentMean` are published at lines 104–105 and are read downstream by `risk_adjustment_analyzer.js` lines 1203 and 1239.

2026-08 changes

None. This file contains no dated change block of any kind. Its filesystem timestamp is 2026-08-30 10:41 with no accompanying comment.

Notes

- `constructor(dataProcessor, …)` immediately does `this.data = dataProcessor || []` — the parameter is an array, not a data processor, and `this.dataProcessor` ends up holding that array. Harmless, but the naming is misleading.
- `findColumnIndex` matches by lower-cased **substring** (`header.includes(name)`), and `analyzeExpenditureMetrics` looks up each year with `findColumnIndex([year])`. Any header containing the string `'2010'` will be taken as the 2010 column. Likewise `findInstitutionData` searches `['unitid','unit_id','id']` — the bare `'id'` will match a header such as `'C21BASIC'` if no earlier candidate hits.
- `assessDataQuality` divides by a hard-coded `totalYears = 14`, which matches this analyzer's 2010–2023 span.
- Comment/code mismatch on the point ladder (see **Scoring**, component 1).
- A stable series is penalised 1 point and a decline of exactly 2 points is penalised 1 point.
- Peer data is gathered, reported and published but never scored.
- `riskFramework.expenditureThresholds.excellent` is dead.

Instructional Support

OPERATING PERFORMANCE & COST STRUCTURE

- **Class**: `UnifiedInstructionalExpenditureAnalyzer` (real analyzer, `unified_instructional_expenditure_analyzer.js`); `UniversalInstructionalExpenditureAnalyzer` (wrapper, `modular_tool_v48 no data cap.html` lines 1548–1581). Orchestrator key `instructionalexpenditure`.
- **Data key**: `'instructional_expenditure'`, fetched by the wrapper at line 1564. Resolved in `data_processing_pipeline.js` line 18 by `/annualized.*pct.*instructional.*expenditure/i`. Declared `dataRequirements: ['annualized pct instructional expenditure.xls']`.
- **What it measures**: instruction as a percentage of total expenditures at the latest available year, plus the change since the earliest available year. The header states the direction: *"Lower instructional expenditure = BAD (quality risk) … Too high instructional expenditure = BAD (inefficiency) … Decreasing trend = BAD (declining investment)"*. `analysisYears` is 2010–2024, `currentYear` 2024.

Sector differences

Two ternaries, at lines 48 and 64, both (`this.sector === '1' || this.sector === '4'`): public and community colleges share one threshold set, everything else (sectors `'2'`, `'3'`, unrecognised) shares the other. A third split at line 454 sets a recommendation-only `optimalRange` of `[55, 75]` public/CC versus `[60, 80]` otherwise.

Scoring

Component 1 — current level (lines 323–362). Evaluated only if `currentRate !== null`. The inefficiency test is checked **first**; the quality ladder follows. First match wins.

#	Threshold key	Public / CC	Private and all others	Test	Floor / ceiling	Points
1	`inefficiencyThresholds.wasteful`	85	90	`currentRate >= threshold`	**floor**	**2**
2	`dangerous`	35	40	`currentRate <= threshold`	**ceiling**	**6**
3	`critical`	45	50	`currentRate <= threshold`	**ceiling**	**5**
4	`concerning`	55	60	`currentRate <= threshold`	**ceiling**	**3**
5	`good`	65	70	`currentRate <= threshold`	**ceiling**	**1**
6	—	—	—	otherwise	—	**0**

`excellent` (75 public / 80 private) is defined and **never read**. Because branch 2 is a ceiling test at 35 (public) / 40 (private), the resulting bands are:

Public / CC current rate	Points	Private current rate	Points
≥ 85	2	≥ 90	2
≤ 35	6	≤ 40	6
> 35 and ≤ 45	5	> 40 and ≤ 50	5
> 45 and ≤ 55	3	> 50 and ≤ 60	3
> 55 and ≤ 65	1	> 60 and ≤ 70	1
> 65 and < 85	0	> 70 and < 90	0

This is the shape that produces the near-constant behaviour recorded in the baseline. The maximum level score, 6 of a 6-point measure, is awarded by a single ceiling test at 35% (public/CC) or 40% (private) — everything at or below that line receives full marks and the ladder does not discriminate below it at all. Zero points require an instructional share **above 65%** of total expenditure (public/CC) or **above 70%** (private). The only downward adjustment available is the peer-median relief below, worth −2 and available to at most the upper half of any peer group. So an institution group whose instructional shares sit at or below the `dangerous` ceiling scores 6, or 4 after relief, on component 1, and the trend component then pushes the

total to the 6-point cap for anything that is not increasing by more than 2 points. Nothing else in the code can move the number.

Peer-median relief (lines 353–359). Applied after the ladder, before the trend:

Condition	Effect
`peerAnalysis?.statistics?.currentRate?.median != null` **and** `currentRate >= peerMedian`	`currentLevelPoints = Math.max(0, currentLevelPoints - 2)`

`calculateStatistics` returns `{ count: 0 }` with no `median` key when the peer array is empty, so `undefined != null` is false and the relief is skipped. This is the **only** one of the three expenditure analyzers whose `calculateStatistics` computes a median (line 579, a true median averaging the two middle values for even n) and the only one whose score depends on peers at all.

Component 2 — trend (lines 365–386). Evaluated only if `totalChange !== null`:

#	Test	Floor / ceiling	Points
1	`totalChange <= trendThresholds.declining` (**−2**)	**ceiling**	**2**
2	`Math.abs(totalChange) <= trendThresholds.stable` (**2**)	band, inclusive	**1**
3	`totalChange >= trendThresholds.improving` (**+2**)	**floor**	**0**
4	otherwise	—	**1**

Branch 1 catches exactly −2. Branch 2 then catches everything in (−2, +2]. Branch 3 is therefore reached only for `totalChange > 2`. **Branch 4 is unreachable**: every real number is covered by branches 1–3. A flat series scores 1, and an increase of exactly +2 scores 1.

`totalPoints = component1 + component2`, then `Math.min(totalPoints, 6)`.

Do the components sum to the declared maximum? No — 6 + 2 = 8 against a declared maximum of 6, so the `Math.min` cap **does** bind. An institution at or below the `dangerous` ceiling with a declining trend scores 8 raw and reports 6.

Maximum points

Declared `this.riskFramework.maxRiskPoints = 6` (line 47); the wrapper re-asserts 6 at line 1578 on the `result.success` path. Framework declares 6 in all four sectors. The wrapper does not set `this.maxRiskPoints`, so the base-class catch path would publish 15.

Reachable, and over-reachable — the raw total can be 8. 6 is attained by any institution scoring 6 on the level (whether or not the trend adds anything), and by several other combinations.

Risk level mapping

`determineRiskLevel(totalRiskPoints)` (lines 595–599), identical to the institutional sibling:

Condition	Level
`totalRiskPoints >= 5`	`High`
`totalRiskPoints >= 3`	`Moderate`
otherwise	`Low`

0–2 Low, 3–4 Moderate, 5–6 High. No `Critical`.

Data-gap behaviour

Identical in structure to the institutional analyzer:

Path	`success`	`riskPoints`	`riskLevel`	`isDataUnavailable`	In denominator?
`catch` in `analyze()`, including *"Institution data not found for UNITID"*	`false`	0	`'Analysis Failed'`	**`true`**	No

Path	success	riskPoints	riskLevel	isDataUnavailable	In denominator?
Wrapper early return, `!institution.unitid`	`false`	3	`'Moderate'`	*not set*; no `maxRiskPoints`	No
Row found, fewer than 2 non-null years	`true`	0	`'Low'`	*not set*	**Yes — 0/6**

No path awards points for missing data.

Peer logic

Same construction as the institutional analyzer: `performPeerAnalysisWithNewFiltering` (lines 221–257) builds its own `StandardPeerBenchmarker`, falls through to the sector-only `performBasicPeerAnalysis` if the benchmarker returns an empty set, and re-reads each peer from the expenditure file, keeping only those with `dataAvailable`.

Is the institution excluded from its own peer group? No — same Number-versus-String mismatch as the institutional analyzer: `extractAllInstitutions()` line 152 stores `unitId: this.parseNumericValue(...)`, the subject is `{ unitId: unitid, unitid: unitid }` with a String, and the benchmarker compares with `!==`. **This one matters for the score**, because the institution's own value is inside the array from which `currentRate.median` is computed, and that median gates the −2 relief.

Statistics. `calculateStatistics` (lines 569–587) returns `{ count, mean, median, values }`. `mean` is arithmetic; `median` is a true median. Scoring reads **median**; reporting reads **mean**. `performExpenditureBenchmarking` calls `calculatePeerRanking(..., true)` (higher is better) for both current level and change, suppressed below 3 valid peers, with the same rank/percentile denominator mismatch noted for the institutional analyzer.

`analysisResults.currentRate` and `peerCurrentMean` (lines 110–111) are read downstream by `risk_adjustment_analyzer.js` lines 1200 and 1238.

2026-08 changes

None. No dated change block in this file. Filesystem timestamp 2026-08-30 10:41 with no accompanying comment.

Notes

- The single ceiling test at 35% / 40% that awards the full 6 points is the whole explanation for the measure behaving as a near-constant; see **Scoring**.
- `assessDataQuality` divides by a hard-coded `totalYears = 14`, but this analyzer collects **15** years (2010–2024), so `dataQuality` can exceed 1.0 and the `> 0.7` → `'High'` banding is computed against the wrong denominator.
- Trend branch 4 is dead code.
- The institution is inside its own peer median; see **Peer logic**.
- Same substring `findColumnIndex` and bare `'id'` lookup caveats as the institutional analyzer.
- `excellent` threshold is dead.
- The class header describes the template as *"REWRITTEN TO MATCH INSTITUTIONAL EXPENDITURE ANALYZER PATTERN"* while the institutional file's header describes itself as an *"EXACT COPY OF INSTRUCTIONAL EXPENDITURE ANALYZER"* — the two files each name the other as their source.

Net Asset Restriction Profile

LIQUIDITY & BALANCE SHEET

- **Class**: `UnifiedRestrictionProfileAnalyzer` (wrapper: `UniversalRestrictionProfileAnalyzer`, wrapper label `'Restriction Profile Analysis'`, analyzer key `restrictionprofile`)
- **Data key**: Two. The wrapper (lines 1140–1141) calls `getDataForAnalyzer('restricted_net_assets')` → `/annualized.*restricted.*net.*asset.*pv.*(and|&).*pu/i` and `getDataForAnalyzer('unrestricted_net_assets')` → `/annualized.*unrestricted.*net.*asset/i`, passed as `{ restricted, unrestricted }`.
- **What it measures**: The share of total net assets that is donor-restricted and therefore unavailable for general operations or debt service — the file header calls this the *"fragile wealth"* pattern, *"where high apparent net assets mask thin operating flexibility."* Inputs named in the header: F2A05 (private) or F1A15+F1A16 (public) for restricted; F2A04 (private) or F1A17 (public) for unrestricted, 2010–2024. It also exposes `restriction_pct` and `expendable_equity_exposed` for Tier 3 consumers (viability / CFI / capitalization).
- **Sector differences**: `measureConfig.maxRiskPoints = (this.sector === '2') ? 6 : 0`. **Only private (sector '2') is scored.** For every other sector `calculateRiskPoints` returns immediately with `riskPoints: 0, maxRiskPoints: 0, riskLevel: 'Informational', levelPoints: 0, levelLabel: 'N/A', driftPoints: 0, informational: true`, and `analysisResults.isInformationalOnly` is set to `this.sector !== '2'`. `analysis_orchestrator.js` instantiates the analyzer only for sectors '1', '2' and '4' (*"Private sector scores; publics and CCs are informational only"*). `risk_scoring_engine.js` declares public 0, private 6, community college 0; there is **no** entry in the `privateForProfit` block. Sector filtering of the input data is its own map: sector '2' → rows with SECTOR in `['2','5']`; sector '1' → `['1','4']`; sector '4' → `['1','4']`; anything else → `[sector]`.
- **Scoring** (private only):

Level component (0–4), on `currentRestrictionPct` expressed as a fraction. Thresholds are **floors** (>=), evaluated in declared order, first match wins.

Condition	Points	levelLabel
`pct >= 0.85`	4	`Critical`
`pct >= 0.75`	3	`High`
`pct >= 0.65`	2	`Moderate-High`
`pct >= 0.50`	1	`Moderate`
`pct >= 0`	0	`Low`
`currentNegativeEquity` (branch taken first)	4 (`maxLevelPoints`)	`Critical`
`pct` null/undefined/NaN	0 (not scored)	`Unknown`

Drift component (0–2), on `driftPP` = `latest.restrictionPct - earliest.restrictionPct` (a fraction, so 0.15 = 15 percentage points). Thresholds are **floors** (>=).

Condition	Points
`drift >= 0.15`	2
`drift >= 0.05`	1
`drift >= -Infinity` (otherwise)	0
`drift` null/undefined/NaN	0 (not scored)

Total:

```
totalPoints = currentNegativeEquity ? maxPoints            // 6, unconditionally
                                    : Math.min(levelPoints + driftPoints, maxPoints)
```

On the normal path the components sum to 4 + 2 = 6, which **is** the declared maximum, so `Math.min` is a no-op. On the negative-equity path the components (4 + 0) do **not** sum to the published total (6) — the total is assigned directly.

- **Maximum points**: `measureConfig.maxRiskPoints` — 6 for sector '2', 0 for all others. **Reachable** for private: either 4 + 2 on the normal path, or the negative-equity branch which assigns 6 outright. For every other sector the maximum is 0, so it is trivially reached and the measure contributes nothing.
- **Risk level mapping**: from `totalPoints`, in `calculateRiskPoints`:

Condition	riskLevel
`totalPoints >= 5`	`Critical`
`totalPoints >= 3`	`High`
`totalPoints >= 1`	`Moderate`
otherwise (0)	`Low`

Non-private sectors bypass this entirely and return `'Informational'`.

- **Data-gap behaviour**: Three distinct paths. 1. Missing unitId, missing UNITID column, institution absent from either source, or no usable year → `createErrorResult` → `riskPoints: 0`, `maxRiskPoints: this.measureConfig.maxRiskPoints` (6 private / 0 otherwise), `riskLevel: 'Unknown'`, `isDataUnavailable: true`. 2. A computable series exists but `currentRestrictionPct` is null/NaN → the level component is skipped with `levelLabel: 'Unknown'` and 0 points; drift likewise skipped if null. `isDataUnavailable` is **not** set on this path. 3. Non-private sectors → `riskLevel: 'Informational'`, `riskPoints: 0`, `maxRiskPoints: 0`, `isDataUnavailable` **not** set.

No path awards points for a genuine data gap. Negative equity is explicitly *not* treated as a data gap — it awards the full maximum. The wrapper's missing-institution early return publishes `riskPoints: 0, riskLevel: 'Unknown'` with no `isDataUnavailable` (the only one of the six wrappers that returns 0 points there).

- **Peer logic**: `findPeerGroup(unitId, peerCriteria)` **ignores `peerCriteria` entirely**. It looks the institution up in `this.institutions` and returns every other institution with the **same `sector` string and the same `carnegie` string** (`i.unitId !== inst.unitId && i.sector === inst.sector && i.carnegie === inst.carnegie`). The institution **is** excluded from its own peer group. Peer statistics are computed over each peer's `currentRestrictionPct` and expose `mean`, `median`, `percentile25`, `percentile75` and the sorted `values`; percentiles use linear interpolation between the two bracketing order statistics. The **report** quotes the **median**. Peer statistics are **not consulted by the scorer at all** — `calculateRiskPoints` receives `peerAnalysis` and never reads it. This measure is scored on absolute thresholds only.
- **2026-08 changes**: seven blocks.
  - 2026-08-28 (v49.1), `extractRestrictionProfile` (line 217) — **negative net assets no longer score as healthy**. What was wrong: *"the year loop skipped only `totalNA === 0` and never rejected a negative total or a negative component, so `restVal / totalNA` escaped [0,1] whenever unrestricted net assets were negative enough. A negative restriction percentage then failed every levelThreshold test down to `{ min: 0 }`, so levelPoints stayed 0 and levelLabel stayed 'Low'."* Evidence: *"5 privates with negative TOTAL net assets score 0 / 'Low' — the balance-sheet failure case scoring as the healthy case — and 32 more produce a restriction percentage outside [0,1]."* Now: *"any year with totalNA <= 0 or restVal < 0 is excluded from the ratio series and recorded instead… A ratio that still lands outside [0,1] (restricted exceeding a positive total, i.e. negative unrestricted) is clamped to [0,1] and flagged via `outOfRange`… Years excluded purely as unreadable data (NaN) remain a data gap and score nothing."*
  - 2026-08-28 (v49.1), line 316 — *"when the most recent observation is a negative-equity year, that IS the current position; report it as such and leave the ratio undefined rather than quoting a stale positive one."* (`currentRestrictionPct` and `driftPP` are nulled.)
  - 2026-08-28 (v49.1), line 152 — `negative_equity` exposed on `analysisResults` *"so Tier 3 consumers can tell a negative-equity institution (ratio undefined, scored at max level severity) from a genuinely low-restriction one."*
  - 2026-08-28 (v49.1), line 400 — the negative-equity branch in `calculateRiskPoints`: *"It now takes the MAXIMUM level points explicitly."*
  - **2026-08-29 (v49.1), line 445 — negative equity now scores the measure's FULL maximum, not just the level maximum**: *"The 28 August change correctly gave the negative-equity case maximum LEVEL points and labelled it Critical. But the same change nulls driftPP - rightly, since there is no ratio from which to compute drift - so the drift component contributed 0 and the total came to 4 of 6. The overall riskLevel needs 5 for Critical, so the institution reported 'High' beside a level label reading 'Critical' and a narrative saying 'scored at the maximum level severity'. An institution whose liabilities exceed its assets is at the top of this measure's scale by definition. It should not be discounted for the absence of a second component that cannot exist in that state."*

- 2026-08-28 (v49.1), lines 476 and 520 — report and narrative made null-safe for the negative-equity case: *"every percentage read here is null-safe instead of printing '0.0%'"*; *"negative equity has no restriction ratio to narrate; it gets its own sentence instead of the 'balanced profile' boilerplate it used to fall through to."*
- 2026-08-28 (v49.1), line 568 — `formatPct` made null-safe: *"printing '0.0%' there would repeat the original bug in the narrative even though the score is now correct."*

- **Notes**:
  - On the negative-equity path the report still prints the arithmetic breakdown "`Risk scoring: {levelPoints} level pts ({levelLabel}) + {driftPoints} drift pts = {riskPoints}/{maxRiskPoints}`", which renders as **"4 level pts (Critical) + 0 drift pts = 6/6"**. The printed sum does not add up.
  - `currentNegativeEquity` is `latestNeg.year >= latest.year` where `latest` is the most recent *ratio* year. A year cannot appear in both lists, so the comparison is effectively strictly greater-than.
  - `outOfRange` can only ever fire on the `> 1` side: the guard above it already rejects `totalNA <= 0` and `restVal < 0`, so `rawPct` cannot be negative.
  - `this.data` (used for `extractAllInstitutions`, and therefore for the peer pool) is `restrictedData` when it has more than one row, otherwise `unrestrictedData`.
  - The header comment's stated scoring (`<50% → 0, 50-65% → 1, 65-75% → 2, 75-85% → 3, ≥85% → 4`; drift `>15pp → 2, 5-15pp → 1`; levels `0=Low, 1-2=Moderate, 3-4=High, 5-6=Critical`) matches the code as written.

Net Revenue Ratio

OPERATING PERFORMANCE & COST STRUCTURE

- **Class**: `UnifiedNetRevenueRatioAnalyzer`
- **Data key**: `'net_revenue_ratio'` (wrapper `UniversalNetRevenueRatioAnalyzer`, HTML line 1284). Internal `this.measureName = 'net_revenue_ratio'`. Wrapper label string is `'Net Revenue Ratio Analysis'`. Dataset regex (`data_processing_pipeline.js` line 52): `/annualized.*net.*revenue.*ratios/i`. Declared `dataRequirements`: `['annualized net revenue ratios.xls']`.
- **What it measures**: Header comment: "Formula: (F1B27 - F1C191) / F1B27. Measures recurring operating surplus as a share of operating + nonoperating revenues. Excludes capital additions (F1B20-F1B23) to isolate recurring operations from one-time capital gifts, grants, and endowment additions. Positive values = recurring surplus, negative values = recurring deficit. Higher values = stronger recurring operating performance." The in-body comment at line 444 adds the private formula: "The source query builds this ratio as ([F1B27]-[F1C191])/[F1B27] for publics and ([F2B01]-[F2B02])/[F2B01] for privates — a decimal, with no *100." The analyzer reads pre-computed year columns matching `/^(19|20)\d{2}$/` (header says 2010–2023); it does not compute the ratio.
- **CONFIRMED — declared at 0 points and reported as "Informational". What it actually does:**

The analyzer itself is **fully scored internally**. `measureConfig.maxRiskPoints = 10`, and `assessMeasureRisk` runs a complete five-factor ladder producing 0–10 points and a real Low/Moderate/High/Critical label. The HTML wrapper then **raises** the reported maximum, not lowers it: `if (result.success && result.analysisResults) { result.analysisResults.maxRiskPoints = 10; }` under the comment "Override max points display".

The zeroing happens only in the framework and the orchestrator. `risk_scoring_engine.js` sets `UniversalNetRevenueRatioAnalyzer: { points: 0 }` in **all four** sector blocks (lines 52, 97, 129, 161); `framework.json` agrees (public 0, private 0, privateForProfit 0, communityCollege 0). The orchestrator's `isExcluded` branch then overwrites `riskPoints → 0`, `maxRiskPoints → 0`, `riskLevel → 'Informational'`, and adds `scoringDisabled: true`.

The framework comment, dated **2026-08-26**, is authoritative on why and is reproduced verbatim:

> *"UNSCORED. 'annualized net revenue ratios.xlsx' and 'annualized adjusted operating margin.xlsx' are the same series: 95.9% of 42,078 shared UNITID x year cells are identical to 4dp (private 99.2%, public 93.6%, CC 94.5%). Scoring both charged one quantity twice — 18 of ~240 framework points. NRR is zeroed rather than AOM because nothing in risk_adjustment_analyzer.js reads netrevenueratio (0 references), while AOM feeds 8 gates (checkFinancialDistress, checkSeverityCondition, checkA5/A8/A9, calculateP5/P9, checkCriticalMassEscalation) — zeroing AOM would silently disable them. The analyzer still runs and reports rate, trend and peer comparison. RESTORE to 10 only after the source query is rebuilt to the documented definition — (F1B27-F1C191)/F1B27 for publics, (F2B01-F2B02)/F2B01 for privates, including nonoperating revenue and excluding capital additions — and this file measurably differs from the AOM file."*

The 2026-08-28 orchestrator comment (quoted in the cross-cutting section) names this measure specifically as the reason the `'Informational'` relabel was added: before it, 441 institutions showed `'Critical'` and 224 `'High'` while contributing 0.0/0.0.

Net effect for documentation: Net Revenue Ratio contributes zero points in every sector; its displayed level is always `'Informational'`; its internally computed level is discarded; its `riskFactors`, `detailedReport` (current rate, 3-year rolling average, trend, long-term change, peer comparison) survive and are displayed.

- **Sector differences**: **none in scoring.** `assessMeasureRisk`, `assessNetRevenueStatus` and every threshold are sector-independent. `this.sector` is used only for `sectorName` (report labels), for `extractAllInstitutions` (peers are restricted to rows whose `SECTOR` cell equals `this.sector`, via `if (this.sector && sector !== this.sector) continue;`), and to construct `StandardPeerBenchmarker`. Framework points are 0 for public, private, privateForProfit and communityCollege alike.
- **Scoring** (as the analyzer computes it, before the framework zeroing):

`maxPoints = this.measureConfig.maxRiskPoints = 10`. Points are fractional (0.25, 0.5, 1.5 all occur). The returned `riskPoints` is `Math.min(riskPoints, maxPoints)`; the raw factor sum maxima are 2 + 4 + 2 + 1 + 1 = **10**, so the clamp never binds.

All value tests route through `assessNetRevenueStatus(value)`, which reads `measureConfig.riskThresholds = { critical: -5.0, moderate: 0.0, adequate: 3.0, excellent: 3.0 }`:

Status	Condition	Direction
`'critical'`	`value < -5.0`	ceiling
`'concerning'`	`-5.0 <= value < 0.0`	ceiling

Status	Condition	Direction
'adequate'	0.0 <= value < 3.0	ceiling
'excellent'	value >= 3.0	floor
'unknown'	value null or undefined	—

Note the status name 'moderate' in riskThresholds maps to the returned label 'concerning', and adequate and excellent are both 3.0, so the excellent key is redundant.

Factor 1 — current-year value (declared 20% weight = 2 points).

Status	Points	Factor string
critical (< -5.0)	2	Critical operating deficit
concerning (< 0.0)	1.5	Concerning operating performance - low/negative surplus
adequate (< 3.0)	0.25	Adequate operating surplus
excellent (>= 3.0)	0	—

The 0.25 carries the in-line annotation "CHANGE THIS: Reduce from 0.5 to 0.25 for positive but low margins" and "← CHANGED FROM 0.5".

Factor 2 — 3-year rolling average (declared 40% weight = 4 points).

Status	Points	Factor string
critical (< -5.0)	4	3-year average shows persistent operating deficits
concerning (< 0.0)	2	3-year average shows weak operating performance
adequate (< 3.0)	0.5	3-year average shows adequate surplus
excellent (>= 3.0)	0	—

Annotations: "CHANGE THIS: Reduce from 3 to 2 points for near-breakeven performance" / "← CHANGED FROM 3"; and on the adequate band, "← CHANGED FROM 1".

Factor 3 — trend direction (declared 20% weight = 2 points). trend is set in analyzeMeasureData from absoluteChange = newest - oldest: > 2.0 → 'improving', < -2.0 → 'declining', otherwise 'stable'.

Condition	Points	Factor string
trend === 'declining'	2	Declining net revenue ratio trend
trend === 'stable'	0.5	Stable net revenue ratio trend
trend === 'improving'	0	—

Note: 'stable' is also the default value assigned when there are fewer than 2 years, so a single-year institution takes the 0.5-point branch.

Factor 4 — volatility (declared 10% weight = 1 point). volatility is the population standard deviation of the scaled year values. Floors.

Condition	Direction	Points	Factor string
volatility > 5.0	floor (strict)	1	High volatility in operating performance
> 3.0 (and ≤ 5.0)	floor (strict)	0.5	Moderate volatility in operating performance
<= 3.0	—	0	—

Factor 5 — peer comparison modifier (declared 10% weight = 1 point). Requires peerAnalysis.success && benchmarking.currentPerformance and a non-null percentile.

Condition	Direction	Points	Factor string
percentile < 25	ceiling	1	Bottom quartile compared to peers

Condition	Direction	Points	Factor string
`25 <= percentile < 50`	ceiling	0.5	Below median compared to peers
`>= 50`	—	0	—

- **Maximum points**: `this.measureConfig.maxRiskPoints = 10`, returned as `maxPoints` and re-asserted as `maxRiskPoints = 10` by the HTML wrapper. Framework: **0** in all four sectors, and the orchestrator's `finalMax` therefore resolves to 0.
- **Risk level mapping**: derived from **`riskPercentage`, a percentage of `maxPoints`**, computed from the **unclamped** `riskPoints` (the clamp is applied only in the returned object):

```
const riskPercentage = (riskPoints / maxPoints) * 100;
if (riskPercentage >= 65) riskLevel = 'Critical';
else if (riskPercentage >= 45) riskLevel = 'High';
else if (riskPercentage >= 25) riskLevel = 'Moderate';
else riskLevel = 'Low';
```

All floors. At `maxPoints = 10`:

Risk level	Percentage floor	Points floor
Critical	≥ 65%	≥ 6.5
High	≥ 45%	≥ 4.5
Moderate	≥ 25%	≥ 2.5
Low	—	< 2.5

These cut-points are NOT the shared 0.80 / 0.55 / 0.30 margin vocabulary. The 2026-08-28 alignment was applied to the three margin analyzers only; Net Revenue Ratio retains 0.65 / 0.45 / 0.25 and no comment in this file references the shared vocabulary. Whatever level this produces is overwritten with `'Informational'` by the orchestrator. The returned `riskPercentage` field is `Math.round((Math.min(riskPoints, maxPoints) / maxPoints) * 100)` — rounded and clamped — which is not the same expression used to pick the level.

- **Data-gap behaviour**: two paths, both flagged.
  - `createFallbackResult(unitId, institutionName, errorReason)` — institution row not found, or `analyzeMeasureData` reports `dataAvailable: false` (no year columns, or no parseable values):

```
{ success: false, measure: 'net_revenue_ratio', institution: {...},
  analysisResults: { currentValue: null, riskPoints: 0, maxRiskPoints: 10,
                     riskLevel: 'Unknown', isDataUnavailable: true },
  detailedReport: {...},
  peerGroup: { criteria: {}, count: 0,
               analysis: 'Peer analysis not performed due to insufficient data' } }
```

  - `createErrorResult(errorCode, errorMessage)` — missing `unitId` (`'MISSING_UNITID'`) or any thrown error (`'ANALYSIS_ERROR'`):

```
{ success: false, measure: 'net_revenue_ratio', error: errorCode, message: errorMessage,
  analysisResults: { riskPoints: 0, maxRiskPoints: 10, riskLevel: 'Unknown',
                     isDataUnavailable: true },
  detailedReport: {...} }
```

`riskLevel` is the string `'Unknown'` and `isDataUnavailable` **is** set (`true`) on **both** paths — this analyzer is the most consistent of the five in this respect. Note `success: false` on both, unlike the margin analyzers' `success: true` fallbacks. The HTML wrapper adds a third path for a missing/invalid institution object: `{ success: false, analysisResults: { riskPoints: 0, maxRiskPoints: 10, riskLevel: 'Unknown' }, detailedReport: { headline: 'Institution data missing for net revenue ratio analysis' } }` — no `isDataUnavailable`. Note that the orchestrator's `isExcluded` relabel to `'Informational'` is applied to any result with an `analysisResults` object, so a data-gap result on this measure is also shown as `'Informational'`, not `'Unknown'`.

- **Peer logic**: this is the **only one of the five** that uses the shared benchmarker. `performPeerAnalysisWithNewFiltering` instantiates `StandardPeerBenchmarker(this.institutions, this.sector, this.sectorName)` when that class is defined, and calls `benchmarker.findAndAnalyzePeerGroup(institution, peerCriteria)`. In `standard_peer_benchmarker.js` the filter order is: self-exclusion by `unitId`/`unitid`; Carnegie (`peerCriteria.carnegieBasic` array membership, applied only when the array is non-empty); HBCU (`peerCriteria.hbcuFilter`, skipped when the value is `'all'`);

custom peer group (UNITID whitelist). If `StandardPeerBenchmarker` is undefined the analyzer falls back to `basicPeerFiltering`, which applies the same four filters inline (and likewise skips the HBCU filter when the value is `'all'` — a check the margin analyzers' hand-rolled filters do not make). Statistics: `calculateStatistics` returns count, **mean**, **median** and the sorted value array for each of `currentValue`, `recent3YearAverage` and `longTermChange`. The **scored** comparison (Factor 5) is a **percentile** — `benchmarkValue` computes `Math.round((peers strictly below / peer count) * 100)`, banded `>= 75` Top quartile/Excellent, `>= 50` Above median/Good, `>= 25` Below median/Fair, else Bottom quartile/**Concerning** (this analyzer uses 'Concerning' where the margin analyzers use 'Poor'). Report findings quote the peer **mean**; `generatePeerComparison` quotes both mean and median.

- **Notes**:
  - **`detectDecimalScale` is per-dataset, not per-row** — the opposite of the (withdrawn) per-row approach in the margin files. Its comment: "Returns 100 when the source file holds decimals, 1 when it already holds percentage points. Decided once per dataset, not per institution: a single institution can legitimately sit under 1% in percentage terms, so a per-row heuristic would misfire on near-breakeven cases. Across a full cohort, some institutions always exceed 1 percentage point." It takes the **90th percentile** of `|value|` across every year cell in the file, not the max: "This ratio is bounded above at 1.0 but unbounded below, so a single institution with expenses above 2x revenue makes max |value| exceed 1.0 even on a decimal file." Scale is 100 if that 90th-percentile absolute value is `< 1.0`, else 1.
  - The reason scaling was added at all (comment at line 444): "Every threshold in measureConfig.riskThresholds is written in percentage points (-5.0, 0.0, 3.0), as is the volatility test in Factor 4 (>3.0, >5.0). Unscaled, the whole cohort collapsed between -5 and +3 and the measure became a sign test: San Marcos at a true -22% scored 'concerning' rather than 'critical', and Factor 4 could never fire at any institution."
  - **Two different rolling-average computations coexist.** The keyed `rollingAverages` map divides by 3 unconditionally (`(yearData[y1] + yearData[y2] + yearData[y3]) / 3`) and will produce `NaN` if any of the three years is missing from `yearData`; `recent3Year` — the value actually scored in Factor 2 — likewise divides the last three **available** values by 3 and requires `years.length >= 3`, else it is `null`. Neither filters nulls the way the margin analyzers' `calculateRollingAverages` does (those require ≥ 2 non-null of 3 and divide by the count present). The rolling-average window labels here are also a different format (`` `${year1}-${year3.toString().slice(-2)}` ``, e.g. `'2021-23'`).
  - The `analysisResults` object published by this analyzer includes `riskPercentage`, which none of the other four publishes.
  - `findColumnIndex` here uses **exact case-insensitive equality**, unlike the `includes`-based matching in the margin and net-tuition analyzers.
  - The file exports itself both as `window.UnifiedNetRevenueRatioAnalyzer` and as a CommonJS `module.exports`; it is the only one of the five with a Node export.
  - It has a `__unitRow` index (2026-08-26 performance work) but **no** `__memoRow` peer-series memo, so `analyzePeerMeasureData` re-parses each peer's full series on every institution.

Net Tuition per FTE

REVENUE DEPENDENCE & PRICING

- **Class**: `UnifiedNetTuitionPerFTEAnalyzer`
- **Data key**: `'net_tuition_per_fte'` (wrapper `UniversalNetTuitionPerFTEAnalyzer`, HTML line 1626 — note this wrapper overrides `analyze()` directly rather than `performSectorAnalysis()`). Internal `this.measureName = 'net_tuition_per_fte'`. Wrapper label string is `'Net Tuition Per FTE Analysis'`. Dataset regex (`data_processing_pipeline.js` line 37): `/annualized.*net.*tuition.*rev.*per.*fte/i`. The header names the source file as `"annualized net tuition rev per fte.xls"`.
- **What it measures**: Header comment: "Analyzes net tuition revenue per FTE for universities using proven peer benchmarking: 1. Current net tuition revenue per FTE (2024) vs peer group average; 2. Percent change in net tuition per FTE (2010-2024) vs peer average." The measure is read from year columns; the analyzer tries ten column-name formats per year (`"2010"`, `Tuition_2010`, `tuition_2010`, `NetTuition_2010`, `net_tuition_2010`, `TuitionPerFTE_2010`, `tuition_per_fte_2010`, `NTFTE_2010`, `ntfte_2010`) across 2010–2024. `current2024` is the **last available** value in that scan, not necessarily 2024; `first2010` is the **first available** value, not necessarily 2010. `percentChange = ((current2024 - first2010) / first2010) * 100`, computed only when there are at least 2 values **and `first2010 > 0`**. `trend` (reported, not scored): `> 5` → 'Increasing'; `< -5` → 'Decreasing'; else 'Stable'; default 'Insufficient data'.
- **Sector differences**: the Factor 1 threshold set is chosen in the constructor by `selectedSector === '1' ? {public} : {everything else}`. Sectors `'2'`, `'3'` and `'4'` all use the second set — **community colleges are scored on the private threshold set**. There is no community-college-specific threshold set. Factor 2 (trend) thresholds are the **same for all sectors**. Framework points are 6 in all four sectors, with the community-college entry carrying the comment "Reduced - low NTR is normal" (`risk_scoring_engine.js` line 146).
- **Scoring**:

`maxPoints = this.riskConfig.maxPoints = 6` (declared components: `currentLevel.maxPoints = 4`, `trendAnalysis.maxPoints = 2`). **There is no clamp**; the ladders themselves cannot exceed 6. Points are **not all integers** — Factor 2 can award 0.5.

Factor 1 — current level vs peer average (max 4 points). `percentDifference = ((current2024 - peerAverage) / peerAverage) * 100`, i.e. percent above or below the peer **mean**. All tests are **floors** (`percentDifference >= X`), evaluated top-down; the first match wins, so the final `else` is the "worse than poor" case.

Band	Sector `'1'` (Public)	Sectors `'2'`/`'3'`/`'4'`	Direction	Points
`>= excellent`	`>= 20`	`>= 15`	floor	0
`>= good`	`>= 5`	`>= 0`	floor	1
`>= moderate`	`>= -10`	`>= -15`	floor	2
`>= poor`	`>= -25`	`>= -30`	floor	3
else (below poor)	`< -25`	`< -30`	ceiling	4

Factor strings, in order: "Strong tuition position: X% above peer average" (0 pts); "Moderate tuition premium: X% above peer average (1 risk point)"; "Near peer average tuition: X% difference (2 risk points)"; "Below-market tuition: X% below peer average (3 risk points)"; "Significantly below-market tuition: X% below peer average (4 risk points)".

Missing-data penalty: if `current2024` is null **or** `peerAnalysis.averages.current2024` is falsy, the factor awards **2 points** — "Tuition competitiveness assessment limited by insufficient peer data (2 risk points)". Note the guard is truthiness, so a computed peer average of exactly `0` also takes this branch.

The threshold comments do not match the code. The constructor annotates the public set as `moderate: -10, // Within ±10% of peer average = 2 points` and `poor: -25, // More than 25% below peers = 4 points`. In code, `>= -25` awards **3** points and only `< -25` awards 4. The code is as tabulated above; the comment on `poor` is wrong about the point value at that threshold.

Factor 2 — growth trend 2010–2024 (max 2 points). Floors on `percentChange`, first match wins. Same thresholds for every sector.

Condition	Direction	Points	Factor string
`growth >= 20` (excellent)	floor	0	Strong tuition growth: X% increase 2010-2024
`>= 10` (good)	floor	0.5	Moderate tuition growth: X% increase 2010-2024 (0.5 risk points)

Condition	Direction	Points	Factor string
`>= 0` (`moderate`)	floor	1	Limited tuition growth: X% increase 2010-2024 (1 risk point)
else (`< 0`)	ceiling	2	Declining tuition trend: X% change 2010-2024 (2 risk points)

`thresholds.poor` is declared as `0` in `riskConfig.trendAnalysis` but is **never referenced** in the code; the final band is a bare `else`. Because `moderate` and `poor` are both `0`, this makes no numerical difference.

Missing-data penalty: if `percentChange` is null (fewer than 2 years, or `first2010 <= 0`), the factor awards **1 point** — "Tuition growth assessment limited by insufficient historical data (1 risk point)".

Consequence worth stating: an institution with no usable data at all scores 2 + 1 = **3 points**, which maps to riskLevel `'High'` (see below), not to a data-unavailable state.

- **Maximum points**: `this.riskConfig.maxPoints = 6`, returned as `totalRisk.maxPoints` and published as `analysisResults.maxRiskPoints`. The HTML wrapper sets both `maxRiskPoints: 6` and `maxPoints: 6` and also sets `this.maxRiskPoints = 6` in its constructor. Framework: 6 in all four sectors.
- **Risk level mapping**: derived from **raw total points**, not from a ratio of the maximum and not from the underlying dollar value:

```
const riskLevel = totalPoints === 0 ? 'Low' :
                  totalPoints <= 2 ? 'Moderate' :
                  totalPoints <= 4 ? 'High' : 'Critical';
```

Risk level	Condition	Direction
Low	`totalPoints === 0`	exact equality — only a zero score
Moderate	`0 < totalPoints <= 2`	ceiling
High	`2 < totalPoints <= 4`	ceiling
Critical	`totalPoints > 4`	floor (i.e. 4.5, 5 or 6)

Note that `'Low'` requires **exactly** 0; a score of 0.5 is `'Moderate'`. Half-point scores are reachable via Factor 2.

- **Data-gap behaviour**: `createErrorResult(message)` is returned when `extractTuitionData(...).hasData` is false (UNITID column absent, institution row absent, or no year value parsed for any year), and on any thrown error:

```
{ success: false, message,
  analysisResults: { riskPoints: 0, riskLevel: 'Unknown', currentValue: null },
  detailedReport: { headline: 'Net tuition per FTE analysis unavailable',
                    keyFindings: [message] } }
```

`riskLevel` is the string `'Unknown'`. **`isDataUnavailable` is NOT set anywhere in this file** — the flag does not appear in the analyzer at all. The HTML wrapper materially rewrites this: it always returns `success: true` and whitelists a fixed set of fields, so an inner failure surfaces as `{ success: true, analysisResults: { riskPoints: 0, riskLevel: 'Unknown', maxRiskPoints: 6, maxPoints: 6, ... } }`. `isDataUnavailable` is not in the wrapper's whitelist and could not pass through even if the analyzer set it. The wrapper's whitelist carries the note: "A5 reads percentChange; the analyzer publishes it and this whitelist was dropping it. Do not remove." Separately, when the **dataset** is absent the wrapper returns `{ success: false, analysisResults: { riskPoints: 3, maxRiskPoints: 6, riskLevel: 'Moderate' }, ... }` — a non-zero **3 points** on a data-absence path — and the same 3/6/'Moderate' on a thrown error.

- **Peer logic**: `findAndAnalyzePeerGroup` builds peers from `this.institutions` (rows whose own `sector` cell equals `this.sector`), minus self by normalised `unitId`, then filters by `peerCriteria.carnegieBasic` (array membership on `c21basic`), `peerCriteria.hbcuFilter` (`'hbcu-only'` / `'non-hbcu-only'`) and `peerCriteria.customPeerGroup` (UNITID whitelist). No `StandardPeerBenchmarker`. The comparison is an **arithmetic mean**, not a median and not a percentile: `performPeerBenchmarking` sums each peer's `current2024` and `percentChange` over peers that return `hasData`, and divides by the count of non-null contributions (`averages.current2024`, `averages.percentChange`). Each of the two averages has its own denominator. No percentile or quartile is computed anywhere in this file.
- **Notes**:
    - **2026-08-29 (v49.1) change**, on `peerPercentChange` in the published record:

> *"the peer trend was computed and shown in the report but never published on the record layer 2 reads, so no gate could compare an institution's net-tuition trajectory with its peers'. P2 needs exactly that: 'progressively discounting' is a statement about the TREND relative to peers, not the level."*

The analyzer now publishes `peerPercentChange: peerAnalysis?.averages?.percentChange ?? null`. **This field is not in the HTML wrapper's whitelist** (`riskPoints`, `riskLevel`, `components`, `maxRiskPoints`, `maxPoints`, `currentValue`, `peerAverage`, `net_tuition_per_fte_normalized`, `percentChange`, `trend`), so it is dropped before the record reaches the orchestrator by that path. This extraction reports the discrepancy without resolving it.

- `analysisResults.net_tuition_per_fte_normalized` is `current2024 / peerAverage.current2024`, published only when `peerCount > 0` and the peer average is truthy; otherwise null.
- `analyze()` is **synchronous** here (no `async`), unlike the four other analyzers in this spec. The wrapper `await`s it, which is harmless.
- The analyzer logs an unconditional `console.log('📦 NTR analyzer outputs', {...})` on every successful analysis — it is not gated by `window.__RAT_QUIET`, unlike the constructor logging in the margin analyzers. It also writes `window.__NTR_HEADERS__`, `window.__NTR_ROW__` and `window.__NTR_KEYS__` on every extraction.
- The file appends a self-installing IIFE (lines 586–683) that defines dev-console helpers `ntr()`, `ntrh()`, `ntrk()`, `ntrr()`, `ntrl()` and `ntrhas()` on the global object and prints a banner at load. This is debug scaffolding, not part of the measure.
- `extractTuitionData` uses a linear `this.data.find(...)` per lookup and is called once per peer per institution; this file has **no** `__unitRow` index and **no** `__memoRow` memo, the two 2026-08-26 performance measures added to the margin analyzers.
- `parseNumericValue` here strips only `,` and `$` (not `%`) and applies **no** scaling — values are dollars per FTE, not percentages.

Operating Margin

OPERATING PERFORMANCE & COST STRUCTURE · reported, does not score

- **Class**: `UnifiedOperatingMarginAnalyzer`
- **Data key**: `'operating_margin'` (wrapper `UniversalOperatingMarginAnalyzer`, HTML line 2363). Internal `this.measureName = 'operating_margin'`. Wrapper's own label string is `'Operating Margin Analysis - No Risk Scoring'`. The dataset regex in `data_processing_pipeline.js` line 31 is `/annualized(?!.*adjusted).*operating.*margin.*pv.*(and|&).*pu/i`. Declared `dataRequirements`: `['annualized operating margin pu.xls']`.
- **What it measures**: The report narrative states "Operating margin measures the percentage of revenue remaining after operating expenses, indicating institutional efficiency and financial sustainability." The analyzer does not compute the ratio; it reads pre-computed year columns (*yyyy preferred, bare yyyy fallback) for 2010–2024. The header comment gives the risk direction as: "Higher margins = lower risk … Rolling averages = lower risk … Negative margins = critical risk … Sub-2% margins = moderate risk (insufficient cushion)." `generateDetailedReport` appends a basis note to the headline that qualifies the measure: for sector `'1'` — "[unadjusted GASB basis — excludes appropriations; not comparable to standard margin benchmarks; see Adjusted Operating Margin]"; for every other sector — "[legacy unrestricted-basis construction; superseded by Adjusted Operating Margin]".
- **CONFIRMED — declared at 0 points and reported as "Informational". What it actually does:**

Three separate mechanisms are in play, and they do not all agree.

1. **Inside the analyzer** (`assessMeasureRisk`, line 658): `const maxPoints = this.sector === '4' ? 12 : 0;` If `maxPoints === 0` the method returns immediately with `{ totalRiskPoints: 0, maxRiskPoints: 0, riskLevel: 'Informational', riskFactors: [], riskPercentage: 0 }`. **No threshold in the ladder is evaluated at all** for any sector other than `'4'`. The in-line comment reads: "CCs use Operating Margin as scored (12 pts); 4-year institutions use it as Informational". The constructor's `maxRiskPoints: 12` carries the comment "Scored for CCs; overridden to Informational for 4-yr via framework". For sector `'4'` the full 12-point ladder below **is** computed and a real Low/Moderate/High/Critical label **is** produced.

2. **In the HTML wrapper** (`UniversalOperatingMarginAnalyzer`, HTML line 2364 onward):

```
// OVERRIDE: Force 0/0 for 4-year institutions; CCs get real scored output
if (result.success && result.analysisResults && this.sector !== '4') {
    result.analysisResults.riskPoints = 0;
    result.analysisResults.maxRiskPoints = 0;
    result.analysisResults.riskLevel = 'Informational';
}
```

The wrapper deliberately lets community-college output through unchanged.

3. **In the framework** (`risk_scoring_engine.js`): `UniversalOperatingMarginAnalyzer` is `{ points: 0 }` in **all four** sector blocks — `public` (line 22), `private` (line 63), `privateForProfit` (line 105) and `communityCollege` (line 155). `framework.json` agrees: `public 0, private 0, privateForProfit 0, communityCollege 0`.

Net effect. Because the community-college framework entry is also 0, the orchestrator's `isExcluded` branch (see cross-cutting section above) fires for community colleges too, and overwrites the analyzer's real CC score back to `riskPoints: 0`, `maxRiskPoints: 0`, `riskLevel: 'Informational'`, `scoringDisabled: true`. **Operating Margin therefore contributes zero points to the total in every sector, community colleges included**, notwithstanding the 12-point CC ladder in the analyzer and the wrapper's deliberate CC exemption. The 12-point CC ladder is computed, is reported through `riskFactors` and `detailedReport`, and is then zeroed. No comment anywhere in these files reconciles the CC-scored intent in the analyzer and wrapper with the CC framework entry of 0.

- **Sector differences**:
  - `sectorName`: `'1'` → Public, `'2'` → Private, `'3'` → For-Profit, anything else → Community College.
  - **Scoring branch**: only sector `'4'` reaches the ladder at all. Within the ladder, Factors 1 and 3 each carry three branches (`sector === '4'`, `sector === '1'`, else). **The `'1'` and `else` branches are unreachable**, because any sector other than `'4'` has already returned `'Informational'` at the top of the method. They are reproduced below for completeness and are marked as dead code.
  - Peer group is drawn only from institutions whose own `sector` cell equals `this.sector` (`extractAllInstitutions`), so a CC is compared only to CCs.
- **Scoring**:

`maxPoints = 12` (sector `'4'` only). `riskPoints` is clamped with `Math.min(riskPoints, maxPoints)` before the risk level is derived.

Factor 1 — 3-year rolling average (`rollingAverages.recent3Year`), community college branch. Every row is a *ceiling* test (`value < X`), evaluated top-down; the first match wins. Code comment: "Standard: +2% or better is healthy; any negative = stress".

Condition	Direction	Points	Factor string
`rollingAverage < -15.0`	ceiling	6	Critical 3-year average operating margin (<-15%)
`< -10.0` (and ≥ -15.0)	ceiling	5	Severe 3-year average operating margin (-15% to -10%)
`< -5.0` (and ≥ -10.0)	ceiling	4	Significant 3-year average operating margin stress (-10% to -5%)
`< 0.0` (and ≥ -5.0)	ceiling	3	Moderate 3-year average operating margin stress (-5% to 0%)
`< 2.0` (and ≥ 0.0)	ceiling	2	Below-standard 3-year average operating margin (0% to 2%)
`< 4.0` (and ≥ 2.0)	ceiling	1	Thin 3-year average operating margin (2% to 4%)
`>= 4.0`	floor	0	(no factor; code comment ">= 4% = no risk points (healthy)")

Factor 1 — dead branches (unreachable; sector `'1'` and else).

Sector `'1'` (Public) — dead	Points		Sector else (Private) — dead	Points
`< -2.0`	5		`< -1.0`	5
`< 0.0`	4		`< 1.0`	4
`< 2.0`	2		`< 3.0`	2
`< 4.0`	1		`< 5.0`	1

Factor 2 — peer comparison (max 2 points). Guarded by `if (benchmarking.currentPerformance && rollingAverage !== null)`. In-line comment says "max 2 points - 17% of max".

Condition	Points	Factor string
`quartile === 'Bottom quartile'` (percentile < 25)	2	Operating margin significantly below peer institutions
`quartile === 'Below median'` (25 ≤ percentile < 50)	1	Operating margin below peer institution median
Above median / top quartile	0	—

Factor 3 — current-year value (most recent non-null year), community college branch. Ceiling tests, first match wins. Code comment: "Standard: +2% or better is healthy; any negative = stress".

Condition	Direction	Points	Factor string
`currentValue < -15.0`	ceiling	4	Current year operating margin critical (<-15%)
`< -10.0`	ceiling	3	Current year operating margin severe (-15% to -10%)
`< -5.0`	ceiling	2	Current year operating margin significant stress (-10% to -5%)
`< 0.0`	ceiling	2	Current year operating margin moderate stress (-5% to 0%)
`< 2.0`	ceiling	1	Current year operating margin below standard (0% to 2%)
`>= 2.0`	floor	0	(no factor; comment ">= 2% = no risk points (healthy)")

Note that the -10 to -5 band and the -5 to 0 band both award 2 points; the ladder is not strictly monotonic in distinct values. This is what the code does.

Factor 3 — dead branches (unreachable).

Sector `'1'` (Public) — dead	Points		Sector else (Private) — dead	Points
`< -10.0`	5		`< -15.0`	5
`< -5.0`	4		`< -8.0`	4

Sector '1' (Public) — dead	Points		Sector else (Private) — dead	Points
< -2.0	3		< -3.0	3
< 0.0	2		< 0.0	2
< 2.0	1		< 2.0	1

Factor 4 — trend deterioration (max 2 points, declared). ACTUAL VALUE: 0 points, always. The block is `if (trend === 'declining') { const changeRate = measureAnalysis.trends?.changeRate || 0; if (changeRate < -2.0) +2; else if (changeRate < -1.0) +1; }`. `calculateMeasureTrends` in **this** file returns `{ direction, overallChange, rollingTrend, annualizedChange, dataPoints, timespan }` — it never writes a `changeRate` key. `changeRate` is therefore always `undefined`, `|| 0` coerces it to `0`, and neither `0 < -2.0` nor `0 < -1.0` can hold. **This factor is inert and awards 0 points to every institution.** This is the exact defect the 2026-08-28 comment in the Adjusted Operating Margin analyzer documents and repairs there; **the same repair was not applied to this file.** See Notes.

Maximum actually reachable for a community college: 6 (F1) + 2 (F2) + 4 (F3) + 0 (F4, inert) = **12**, which coincides with the declared cap.

- **Maximum points**: as the analyzer computes it — `this.sector === '4' ? 12 : 0`. Constructor declares `measureConfig.maxRiskPoints = 12`. Framework declares 0 in all four sectors. The orchestrator's `finalMax` is therefore 0 in all four sectors.
- **Risk level mapping**: derived from **points as a ratio of `maxPoints`**, under the 2026-08-28 (v49.1) shared-margin-vocabulary block (quoted in full under Structural Margin below):

```
let riskLevel = 'Low';
if (riskPoints >= maxPoints * 0.80) riskLevel = 'Critical';
else if (riskPoints >= maxPoints * 0.55) riskLevel = 'High';
else if (riskPoints >= maxPoints * 0.30) riskLevel = 'Moderate';
```

All three are **floors** (`>= X`). At `maxPoints = 12` the cut-points are:

Risk level	Ratio floor	Points floor	Integer points
Critical	≥ 0.80	≥ 9.6	10, 11, 12
High	≥ 0.55	≥ 6.6	7, 8, 9
Moderate	≥ 0.30	≥ 3.6	4, 5, 6
Low	—	< 3.6	0, 1, 2, 3

For every non-CC sector the level is the literal string `'Informational'`, set before any arithmetic. For CC, the label computed here is then overwritten to `'Informational'` by the orchestrator's `isExcluded` branch. `riskPercentage = Math.round((riskPoints / maxPoints) * 100)`.

- **Data-gap behaviour**: `createFallbackResult(unitId, institutionName, errorMessage)` is returned when the institution row is not found or when `hasValidData` fails (fewer than 2 non-null years). It returns:

```
{ success: true,
  measure: 'operating_margin',
  institution: { unitId, name },
  analysisResults: { riskPoints: 0, isDataUnavailable: true },
  detailedReport: { headline: '<name> operating margin analysis unavailable', ... } }
```

`isDataUnavailable` **is** set (`true`). `success` is deliberately `true` — the code carries the annotation "CHANGED: false → true" and, on `riskPoints`, "CHANGED: 6 → 0", and on the flag "ADDED: This flag tells the main tool to exclude these points". **`riskLevel` is not set at all** in the fallback object — the key is absent, not `'Unknown'`. Downstream, the orchestrator's `isExcluded` branch writes `riskLevel: 'Informational'` into it. If `institution` has no `unitId`, `analyze()` calls `this.createErrorResult(...)` — **which is not defined on this class** (see Notes).

- **Peer logic**: `findAndAnalyzePeerGroup` builds the peer set from `this.institutions`, which is itself restricted at construction time to rows whose `sector` cell equals `this.sector`. Filters applied, in order: self-exclusion by `unitId`; `peerCriteria.carnegieBasic` (array membership on `c21basic`); `peerCriteria.hbcuFilter` (`'hbcu-only'` keeps `hbcu === '1'`, `'non-hbcu-only'` drops it); `peerCriteria.customPeerGroup` (UNITID whitelist). It does **not** use `StandardPeerBenchmarker`. The scored comparison (Factor 2) is a **percentile**, not a mean or median: `benchmarkValue` computes `percentile = Math.round((count of peers strictly below the institution / peer count) * 100)`, then bands it — `>= 75` Top quartile/Excellent, `>= 50` Above median/Good, `>= 25` Below

median/Fair, else Bottom quartile/Poor. Peer **medians** are computed and used in the narrative findings only (`peerAnalysis.statistics.currentMargin.median`, `.recent3YearAverage.median`); peer **means** are used only by `generatePeerComparison`, which produces the display string "N% above/below peer average".

- **Notes**:
  - **Factor 4 is inert in this file.** The 2026-08-28 (v49.1) comment in `unified_adjusted_operating_margin_analyzer.js` (lines 810–824) documents this bug and its repair — "this factor read `measureAnalysis.trends?.changeRate`, but `calculateMeasureTrends` never writes a `changeRate` key … the factor has been INERT since it was written, awarding 0 of its 2 points to every institution" — and the fix (reading `annualizedChange`) was applied to the Adjusted Operating Margin analyzer only. The identical block in this file still reads `changeRate`. No comment in this file acknowledges it.
  - **`createErrorResult` is called but never defined** on `UnifiedOperatingMarginAnalyzer` (called at lines 198 and 260; no method definition exists in the file). The line-198 call sits inside the `try`, so it throws a `TypeError` that is caught by the `catch` at line 258 — which then calls `createErrorResult` again, from inside the catch, where the second `TypeError` propagates out of `analyze()` uncaught. The same omission exists in the Adjusted Operating Margin analyzer. `UnifiedStructuralMarginAnalyzer` does define it.
  - **`assessRollingAverageTrend` and `calculateRollingAverageTrendChange` read the wrong keys.** They look up `rollingAverages['2010-12']` and `rollingAverages['2021-23']`, but `calculateRollingAverages` writes labels of the form `` `${yearWindow[0]}-${yearWindow[2]}` ``, i.e. `'2010-2012'` and `'2021-2023'`. `assessRollingAverageTrend` therefore always returns `'insufficient-data'`, and `calculateRollingAverageTrendChange` returns `NaN` (both operands are `undefined`, and the guard tests `!== null`, which `undefined` passes). This affects the reported `rollingAverageTrend` field and the unused `rollingAverageTrend` peer statistic; it does **not** affect scoring. The Structural Margin analyzer uses the correct four-digit keys.
  - **No isolated-breach rejection and no per-row scaling.** Unlike the Adjusted Operating Margin analyzer, this file has no `__rejectIsolatedBreaches`, no `rowScale` and no `parseScaled`. Its `parseNumericValue` applies the magnitude test the 2026-08-26 comments elsewhere describe as "not a units test": `if (num > 0 && num < 1) return num * 100;` and `if (num < 0 && num > -1) return num * 100;`, otherwise the value is returned unscaled. The 2026-08-26 comment retained in the sibling files states the operating margin file "is genuinely mixed within the public sector (398 rows stored as decimals, 334 as percent)", and that under this magnitude test "a decimal -4.229 is a -423% margin … it was left unscaled and read as -4.2% -> scored High instead of Critical. 271 of 296 negative mis-scaled cells landed in the wrong band this way."
  - **Performance annotations dated 2026-08-26** (`__unitRow`, `__memoRow`): both are marked "pure performance" and state that they change no score. `__unitRow` is documented as FIRST-WINS "deliberately", because a Map built with unconditional `set()` "would silently change which row is analysed if a dataset ever carried a repeated UNITID. That would be a scoring change."
  - `this.measureName` is `'operating_margin'`, which is **the same string** the Adjusted Operating Margin analyzer sets (see that section).

Pell Grant Recipients

MARKET DEMAND & STUDENT SUCCESS

- **Class**: `SimplePellAnalyzer` (wrapper: `UniversalPellAnalyzer`, wrapper label `'Pell Income Analysis'`, analyzer key `pell`)
- **Data key**: The analyzer does not call `getDataForAnalyzer` itself. The wrapper (lines 2706–2710) calls it five times and passes the results as a `dataSources` object:

Constructor property	Data key	Filename pattern (`data_processing_pipeline.js`)
`federalRisk`	`federal_risk`	`/BBB.*risk.*factor.*pv.*(and|&).*pu/i`
`pellIncomeUnder30k`	`pell_income_under30k`	`/annualized.*percent.*of.*pell.*_30k.*pv.*(&|and).*pu/i`
`pellIncome30to48k`	`pell_income_30to48k`	`/annualized.*percent.*of.*pell.*30-48k.*pv.*(&|and).*pu/i`
`pellIncome48to75k`	`pell_income_48to75k`	`/annualized.*percent.*pell.*48-75k.*pv.*(and|&).*pu/i`
`pellIncome75to110k`	`pell_income_75to110k`	`/annualized.*percent.*pell.*75-110k.*pv.*(and|&).*pu/i`

- **What it measures**: The share of students receiving Pell grants (column `Pgrant_p` in the federal-risk file), scored as exposure to federal aid policy volatility rather than as a performance judgement — the analyzer emits an explicit note to that effect (`getPellPolicyExposureNote()`: *"Higher Pell enrollment is not a negative performance indicator. Risk points reflect exposure to federal aid policy volatility, not mission value or student success."*). The four household-income-band files (under $30k, $30–48k, $48–75k, $75–110k) supply reported levels and 2010-onward trends for the narrative only; **none of the four income bands enters the score.**
- **Sector differences**: Only the Factor 1 threshold pair differs. `sector === '1'` (Public) uses `{ good: 35, poor: 50 }`; **every other sector** — private, for-profit and community college — uses `{ good: 25, poor: 40 }`. The in-code comment labels these two branches "Public" and "Private" only; sectors '3' and '4' silently take the private branch. Factor 2 (peer percentile) has one threshold pair for all sectors. Declared framework points are 4 in all four sector blocks of `risk_scoring_engine.js`. The narrative adds a sector-context sentence with a distinct community-college variant (`this.sector === '4'`).
- **Scoring**:

Factor 1 — Overall Pell dependency (0–2 points). Thresholds are **floors** (>=).

Condition (public, sector '1')	Condition (all other sectors)	Points
`pellPercentage >= 50`	`pellPercentage >= 40`	2
`pellPercentage >= 35`	`pellPercentage >= 25`	1
otherwise	otherwise	0
value is `null`	value is `null`	0 (not scored)

Factor 2 — Peer comparison (0–2 points). Gated on: a non-null peer average, a non-null institution value, **and** `peerValues.length >= 3`. `percentile` is `round(count of peers strictly below the institution / peer count * 100)`.

Condition	Points
`percentile >= 75` (floor)	2
`percentile > 50` (strict floor — 50 itself scores 0)	1
`percentile <= 50`	0
fewer than 3 peer values, or peer average null, or institution value null	0 (not scored)

The two components sum to exactly 4, which **is** the declared maximum. There is no `Math.min` cap in `calculateTotalRisk`; the sum cannot exceed 4 by construction.

- **Maximum points**: `this.riskConfig.maxPoints = 4`. Published as `analysisResults.maxRiskPoints = this.riskConfig.maxPoints`, then overwritten by the wrapper to a literal 4. **Reachable** — 2 + 2 = 4.
- **Risk level mapping**: computed in `calculateTotalRisk` from `totalPoints`:

totalPoints	riskLevel
`=== 0`	`Low`
`<= 2` (i.e. 1 or 2)	`Moderate`
`<= 3` (i.e. 3)	`High`
otherwise (4)	`Critical`

Note there is no `Low` band above zero: a single point publishes `Moderate`.

- **Data-gap behaviour**: `extractPellData` returns `hasData = sourcesFound > 0`. If `hasData` is false, `analyze()` returns `createErrorResult('No Pell grant data found for institution')` → `riskPoints: 0`, `maxRiskPoints: 4`, `riskLevel: 'Unknown'`, `isDataUnavailable: true`, `currentValue: 0`. **No inner path awards points for missing data** — both factors were changed on 2026-08-29 to score 0 when their input is absent. The wrapper's missing-institution early return is the exception: it publishes `riskPoints: 6, maxRiskPoints: 4, riskLevel: 'Moderate'` with no `isDataUnavailable` — 6 points against a declared maximum of 4.
- **Peer logic**: Two paths. If `StandardPeerBenchmarker` is defined (`hasEnhancedPeerFiltering`), the analyzer constructs one over `this.institutions` and calls `benchmarker.findAndAnalyzePeerGroup(institution, peerCriteria)` with the real institution object (changed 2026-08-29 — see below), so the benchmarker's own self-exclusion applies. Otherwise the built-in fallback filters `this.institutions` (already restricted to the caller's sector by `extractAllInstitutions`), excluding `inst.unitId !== institution.unitid`, then applying `peerCriteria.carnegieBasic` (array membership, string-compared) and `peerCriteria.hbcuFilter`. **The fallback path implements no `customPeerGroup` filter**, unlike the other five analyzers. Comparison basis: `averages.overallPell` is an arithmetic **mean** used for the narrative and `calculatePeerRanking`; the **scored** Factor 2 uses a **percentile** against `averages.peerValues` (every peer's `overallPellPercentage`). The institution is excluded from its own peer group on both paths.
- **2026-08 changes**: four blocks, all dated 2026-08-29 (v49.1). 1. `riskConfig.peerComparison` (line 63) — **direction corrected**: *"This factor scored the BOTTOM peer quartile - the LOWEST Pell share - at 2 points, the exact opposite of Factor 1 above, which charges 2 points for a HIGH Pell share as federal policy exposure. The two halves of one measure ran in opposite directions. It never showed, because the factor never executed (see calculateTotalRisk). Now: percentile is the share of peers BELOW this institution, so a high percentile is a high Pell share relative to peers, and that is the risk."* 2. `findAndAnalyzePeerGroup` (line 125) — the `StandardPeerBenchmarker` call was passing a placeholder: *"was {unitId: 'dummy'}, so the benchmarker's self-exclusion matched nothing and the institution sat inside its own peer group. That biased its own percentile toward the middle - and it matters now, because the peer factor above has just been repaired and actually runs. The real institution is in scope here."* 3. `calculateTotalRisk` Factor 1 data gap (line 561): *"was `totalPoints += 1`. A missing Pell figure was charged as risk. Data gaps never score - the house rule stated in the transfer-out and grant-funding headers, and the rule already applied to enrolment on 28 August."* 4. `calculateTotalRisk` Factor 2 peer-value read (line 571): *"was `peerAnalysis.peerValues`. performPeerAnalysis publishes peerValues INSIDE the averages object, so this read was one level too high and resolved to undefined on every institution ever scored. The length check below then failed and this entire factor - 2 of the measure's 4 points - never awarded anything. Pell could reach 2 of 4, and High and Critical were unreachable."* The read is now `peerAnalysis.averages?.peerValues ?? peerAnalysis.peerValues ?? []`.
- **Notes**:
  - An institution with income-band data but **no** `Pgrant_p` value has `hasData === true` (`sourcesFound > 0`) and so passes the error gate, but then scores 0 on both factors and publishes `riskLevel: 'Low'` with **no** `isDataUnavailable`. A data gap presents as the healthy case and keeps its 4 points in the denominator.
  - `performPeerBenchmarking` contains a duplicated accumulation block (lines 308–316): `income30to48kTrends.totalChange` is added to `peerIncome30to48kTrendSum` and `peerIncome30to48kTrendCount` twice per peer. The resulting mean is unchanged (numerator and denominator both double), so this is inert arithmetically.
  - `performPeerBenchmarking` calls `extractPellData` once per peer in the `forEach` and again in the `peerValues` map — every peer's five source files are parsed twice.
  - Year-column matching in `extractPellData` and `extractIncomeDistributionData` uses `header.toString().includes(year.toString())`, a substring test, not an exact match.
  - `calculatePercentile` returns `null` when `peerValues` is empty, but that case is already excluded by the `length >= 3` gate.

- The policy-exposure note is pushed into `findings` **before** the institution's own Pell finding, so it appears first in the report's bullet list.

Primary Reserve Ratio

LIQUIDITY & BALANCE SHEET

- **Class**: `UnifiedPrimaryReserveAnalyzer`
- **Data key**: `'primary_reserves'` (wrapper `UniversalPrimaryReserveAnalyzer`, HTML line 2447). Internal `this.measureName = 'primary_reserve_ratio'`; `dataRequirements = ['annualized primary reserves ratio pu.xls']`. Wrapper label string is `'Primary Reserve Analysis'`.
- **What it measures**: The header comment defines the units as "Percentage (ratio of expendable net assets to total expenses)" and the risk framework as "Higher ratios = lower risk (better liquidity)... Sub-15% ratios = critical risk (insufficient reserves), Sub-25% ratios = moderate risk (limited liquidity cushion)". The report narrative says it "indicates the institution's ability to meet operating expenses from expendable net assets".
- **Sector differences**:
  - `sectorName`: `'1'` Public, `'2'` Private, `'3'` For-Profit, anything else Community College.
  - The ratio component of scoring branches on sector `'1'` OR `'4'` → percentile-relative; every other sector (including `'3'`) → absolute percentage bands. Comment: "Percentile-based status for public sector. GASB 68/75 drives expendable net assets negative sector-wide, so the 15/25/35/50 bands degenerate to a sign test."
  - The trend component (3 points) is identical across all sectors.
  - The peer group is sector-restricted by construction (see Peer logic).
- **Scoring**:

`maxPoints` is hard-coded to `10` inside `assessMeasureRisk`. Values are stored in the file as decimals and multiplied by 100 by `parseNumericValue` ("File stores ratios (1.25 = 125%); thresholds and display are in percent"), so all absolute thresholds below are in percentage points.

Component 1 — 3-year rolling average (7 points max). Guarded by `rollingAverage !== null`.

Public / Community College (sector `'1'` or `'4'`) — `reserveStatusFromPercentile` applied to `benchmarking.rollingPerformance.percentile`:

Percentile band (ceiling)	Status	Points
percentile < 10	critical	7
percentile < 25	weak	5
percentile < 50	fair	3
percentile < 75	good	1
percentile >= 75	strong	0
percentile null / undefined / non-finite	unknown	0

All other sectors — absolute ceilings:

Rolling average (ceiling)	Points	Code label
< 15.0	7	Critical: <15% reserves
< 25.0	5	High: 15-25% reserves
< 35.0	3	Moderate: 25-35% reserves
< 50.0	1	Low: 35-50% reserves
>= 50.0	0	Minimal risk

Component 2 — trend (3 points max). Awarded only when `trends.direction === 'declining'`. `changeRate = trends.overallChange / trends.timespan` (0 when `overallChange` is null or timespan is not > 0).

Condition (ceiling)	Points
changeRate < -5.0	3
changeRate < -2.0	2
otherwise (still declining)	1

Stable or improving trends add 0. Total is `Math.min(riskPoints, 10)`.

`trends.direction` itself: within the window starting at `TREND_START_YEAR`, `overallChange = last – first`; `> 2` improving, `< -2` declining, else stable. Fewer than two in-window observations returns `'insufficient-data'`.

- **Maximum points**: `measureConfig.maxRiskPoints = 10`, and `assessMeasureRisk` independently hard-codes `const maxPoints = 10; // Enhanced framework: 10 points total`. The HTML wrapper also forces `maxRiskPoints = 10` on success, with the comment "Both sectors get 10 points". Component caps 7 + 3 sum exactly to 10.
- **Risk level mapping**: from raw points (floors), not percentage, though the inline comments express them as percentages of max:

Condition	riskLevel	Code comment
riskPoints >= 9	Critical	90%+ of max points
riskPoints >= 7	High	70%+ of max points
riskPoints >= 4	Moderate	40%+ of max points
otherwise	Low	—

- **Data-gap behaviour**: `createFallbackResult` (institution row not found, or "Insufficient primary reserve data") and `createErrorResult` (missing unitId, or a thrown exception) both return `success: false` and `analysisResults: { riskPoints: 0, maxRiskPoints: 10, riskLevel: 'Unknown', isDataUnavailable: true }`. The HTML wrapper returns the same shape when the primary reserves file is absent, with the finding "Please upload: annualized primary reserves ratio pu.xls".
- **Peer logic**: This analyzer uses its **own** `findAndAnalyzePeerGroup`, not `StandardPeerBenchmarker`. `extractAllInstitutions()` keeps only rows whose sector matches `this.sector`; the peer filter then excludes self and applies Carnegie basic, HBCU (`'hbcu-only'` / `'non-hbcu-only'`) and `customPeerGroup` when supplied. Comparison is a **percentile rank**: peers strictly below the institution (`institutionValue > v`, higher-is-better) ÷ peer count, rounded; `>= 75` Top quartile/Excellent, `>= 50` Above median/Good, `>= 25` Below median/Fair, else Bottom quartile/Poor. `calculateStatistics` returns count, mean, median, min, max; the **median** is what the report findings print, while `generatePeerComparison` (used for the `peerComparison` field) is computed against the **mean**.
- **Notes**:
  - `static TREND_START_YEAR = 2019`, with the same GASB 68 / GASB 75 rationale as the viability analyzer: "2019 is the first year clean of both. Move this as the panel extends."
  - The Component-1 guard is `if (rollingAverage !== null)`. `calculateRollingAverages` assigns `recent3Year` only when `years.length >= 3`; otherwise the key is absent, so `rollingAverage` is `undefined`, which passes `!== null`. Every absolute comparison against `undefined` is false, so `ratioPoints` stays 0 and the 7-point component contributes nothing. This is the same failure mode the viability analyzer documents and fixed on 2026-08-28 with `!= null`; no equivalent fix is present in this file.
  - `measureConfig.riskThresholds` is declared as `critical: 15.0, moderate: 25.0, adequate: 50.0, excellent: 50.0` with inline comments reading "25-40% = adequate reserves" and "Above 40% = excellent liquidity". Those comments do not match the declared values, and `assessMeasureRisk` does not read `riskThresholds` at all — it hard-codes 15 / 25 / 35 / 50. `measureConfig.riskWeights` (0.2 / 0.4 / 0.2 / 0.1) is likewise never read.
  - `assessLiquidityStatus` (reported as `liquidityStatus`, not scored): `>= 40.0` excellent, `>= 25.0` good, `>= 15.0` adequate, else concerning; `'unknown'` for null/undefined.
  - `assessRollingAverageTrend` compares hard-coded window keys `'2010-12'` and `'2021-23'`, and the `'2021-23'` key is assigned the most recent 3-year average regardless of which years those actually are. Change `> 2` improving, `< -2` declining, else stable.
  - **2026-08-26** — `__unitRow` UNITID index and `__memoRow` peer-series memo added, both labelled "pure performance". The memo comment states it is "Exact, not approximate. This caches parsing only. Every statistic is still computed per institution against that institution's own peer set, so self-exclusion from the peer group is untouched and scores do not move."
  - A range-validation block in `parseNumericValue` (rejecting values `< -1` or `> 10`) is commented out.

- Header comment says the data source is "annualized primary reserves ratio pu.xls (PUBLIC UNIVERSITIES)", while the data-processing pipeline pattern for `'primary_reserves'` is `/annualized.*primary.*reserves.*ratio.*pv.*(and|&).*pu/i`, i.e. combined pv & pu.

Program Distribution

ACADEMIC PROGRAM BREADTH & SUSTAINABILITY

- **Class**: `UnifiedProgramDistributionAnalyzer` (real analyzer, `unified_program_distribution_analyzer.js`); `UniversalProgramDistributionAnalyzer` (wrapper, `modular_tool_v48 no data cap.html` lines 1515–1547). Orchestrator key `programdistribution`.
- **Data key**: two keys, both fetched by the wrapper (lines 1530–1531), not by the real analyzer: `'program_distribution'` → constructor arg 1, and `'certificate_distribution'` → constructor arg 3. Resolved in `data_processing_pipeline.js` lines 20–21 by `/^(?!.*cert).*concentration.*data/i` and `/concentration.*data.*cert/i`. Declared `dataRequirements: ['concentration data.xls', 'concentration data for certs.xls']`.
- **What it measures**: how concentrated the institution's degree output is across CIP codes, relative to its peer group, at each degree level. Two source columns per level — `Top5 <level>s` (share of degrees in the five largest programmes) and `CSI <level>s` — are standardised against the peer distribution and averaged into a composite the code calls PBI. Per the in-code comment at lines 431–433: *"Composite concentration = mean of the two z-scores. Positive = more concentrated than peers (monoculture direction); negative = more dispersed than peers (sprawl direction)."*

Sector differences

No sector-specific thresholds. `this.sector` selects `sectorName` and is the sector filter inside `StandardPeerBenchmarker` (step 5). What varies is institution type, from `classifyInstitution()` (lines 101–121), which reads `institution.SECTOR || institution.sector` and `institution.C21BASIC || institution.C2BASIC || institution.carnegieClassification`:

Type	Test	`applicableMeasures` (degree levels)	`certApplicability`
`community_college`	sector `'4'` (checked first, Carnegie ignored)	`Top5/CSI AAs`, `Top5/CSI BAs`	`Cert20`, `Cert21`, `Cert2`, `Cert4`
`liberal_arts`	sector `'1'`/`'2'` and C21BASIC in `'21','22'`	`Top5/CSI AAs`, `BAs`, `MAs`	`Cert6`
`university`	sector `'1'`/`'2'` and C21BASIC **not** in `'21','22'`	`Top5/CSI AAs`, `BAs`, `MAs`	all six cert types

Fallthrough (line 120) returns `university`; sector `'3'` always lands there. **Community colleges never score the MA level** because `MA` is not in their `applicableMeasures` and the level loop (line 679) tests `typeConfig.applicableMeasures.some(m => m.includes(level))`.

Scoring

Step 1 — composite per degree level (`calculatePBI`, lines 405–460). For each of AA, BA, MA that is applicable:

- `instTop5 = data['Top5 <level>s']`, `instCSI = data['CSI <level>s']`; returns null if either is null.
- `CSI_IS_DIVERSITY = true` is hard-coded, so `instCSI_adj = 1 - instCSI`.
- Peer arrays (`__getPeerStats`, lines 467–516) collect `peerTop5` and `1 - peerCSI` for every peer with both values present. Returns null if fewer than 2 peers have both.
- `calculateZScore` (lines 613–624) uses the **population** standard deviation (divide by `n`, not `n-1`) and returns 0 when `stdDev === 0` or `peerValues.length < 2`.
- `pbi = (top5ZScore + csiZScore) / 2`.
- `percentile = calculatePercentileRank(pbi, peerPBIs)` (lines 627–632) = `((below + 0.5 * equal) / n) * 100`, returning a flat 50 if fewer than 2 peer PBIs.

Step 2 — discriminating-level gate (lines 734–741). A level may drive the score only if **all three** hold:

Test	Floor / ceiling
`result.peerCount >= 10`	**floor**
`result.peerTop5Median < 0.95`	**ceiling** (`>= 0.95` disqualifies)
`result.peerTop5AtCeiling < 0.40`	**ceiling** (`>= 0.40` disqualifies)

`peerTop5AtCeiling` is the share of peers with `Top5 >= 0.999`. Disqualified levels are recorded in `nonDiscriminatingLevels`, reported in `keyFindings`, and never scored.

Step 3 — pick the worst surviving level (lines 747–761). `extremity = Math.max(result.percentile, 100 - result.percentile)`; the level with the strictly greatest extremity wins (initial `worstExtremity = 0`, so ties go to the first level encountered in AA, BA, MA order).

Step 4 — points (lines 765–771).

Condition	Floor / ceiling	Points
`worstExtremity >= 95`	**floor**	6
`worstExtremity >= 90`	**floor**	3
otherwise	—	0

Because `extremity = max(p, 100 - p)`, "≥ 95" means the institution's PBI percentile within its peer group is ≥ 95 or ≤ 5; "≥ 90" means ≥ 90 or ≤ 10.

Step 5 — direction (lines 774–778). `worstDirection = 'balanced'` unless a worst level exists *and* `basePoints > 0`, in which case `pbiResults[worstLevel].pbi < 0 ? 'sprawl' : 'monoculture'`. A PBI of exactly 0 reads as monoculture.

`totalRiskPoints = basePoints` — line 779 is explicit: *"No add-on; max 6"*. There is no second component; the components do **not** need to sum, because there is only one.

Maximum points

Declared `this.riskFramework.maxRiskPoints = 6` (line 46); `maxPossiblePoints: 6` is hard-coded in all three return paths of `calculateRiskAssessment`; the wrapper sets `this.maxRiskPoints = 6` and overwrites `result.analysisResults.maxRiskPoints = 6` (line 1543). Framework declares 6 in all four sectors.

Reachable — 6 is awarded whenever a discriminating level puts the institution at the outer 5% of its peer group in either direction. Only three values are attainable: **0, 3 and 6**.

Risk level mapping

`determineRiskLevel(riskPoints)` (lines 836–843), `percentage = (riskPoints / 6) * 100`:

Condition	Level	Equivalent points (out of 6)
`percentage <= 25`	`Low`	0 – 1.5
`percentage <= 45`	`Moderate`	> 1.5 – 2.7
`percentage <= 65`	`High`	> 2.7 – 3.9
otherwise	`Critical`	> 3.9 – 6

Since only 0, 3 and 6 are attainable, this measure reports **only `Low`, `High` and `Critical`. The `Moderate` band is unreachable.**

Data-gap behaviour

Path	`success`	`riskPoints`	`riskLevel`	`isDataUnavailable`	In denominator?
`createNoDataResult` — no institution row in the concentration file, or every applicable measure null	`true`	0	`'Unknown'`	**`true`**	No — excluded
`createErrorResult` — any throw inside `analyze()`	`false`	0	`'Error'`	**`true`** (also `isError: true`)	No
`analyzeInstitution(unitid)` test path, institution not found	`false`	0	`'Error'`	**`true`**	No
Wrapper early return, `!institution.unitid`	`false`	0	`'Critical'`	*not set*	No — excluded on `!success`
`peerData.length < 2` (lines 658–670)	`true`	0	`'Low'`	*not set*	**Yes — 0/6**
`validLevels === 0` (lines 701–712)	`true`	0	`'Low'`	*not set*	**Yes — 0/6**

Path	success	riskPoints	riskLevel	isDataUnavailable	In denominator?
Every applicable level non-discriminating → `worstLevel === null`	`true`	0	`'Low'`	*not set*	**Yes — 0/6**

The last three rows all set a `reason` string inside `riskAssessment` and, for the suppressed-levels case, a `keyFindings` line reading *"No discriminating degree level available — program distribution not scored for this institution."* — but `analysisResults` still publishes `riskLevel: 'Low'` with no `isDataUnavailable`, so the measure sits in the denominator at 0 of 6 and reads as a clean pass.

No path awards points for missing data.

Peer logic

`performPeerAnalysis()` (lines 359–372) branches on `this.hasEnhancedPeerFiltering`, which is true whenever `StandardPeerBenchmarker` is defined — it is loaded by the tool, so the **enhanced branch is the live one and `performBasicPeerAnalysis` is effectively dead**.

Enhanced branch: `new StandardPeerBenchmarker(this.institutions, this.sector)` — note `sectorName` is not passed, so the benchmarker's log lines read `undefined`. The criteria object is built as `{...peerCriteria, sectorFilter: this.sector, institutionTypeFilter: this.institutionTypes[institutionType].displayName}`. **`StandardPeerBenchmarker` reads neither `sectorFilter` nor `institutionTypeFilter`** — its filter chain is: self-exclusion, `peerCriteria.carnegieBasic`, `peerCriteria.hbcuFilter`, `peerCriteria.customPeerGroup`, then `inst.sector === this.sector`. So the liberal-arts / university split **is not applied to the peer group**; it only selects which levels are analysed.

Basic branch (lines 374–399): self-exclusion on `peer.unitid === institution.unitid`, custom peer group, then `typeConfig.sectorCheck(peerSector) && typeConfig.carnegieCheck(peerCarnegie)` — note this branch does **not** compare against `this.sector`, only against the type's sector predicate, so `['1','2'].includes(peerSector)` admits publics and privates together.

Is the institution excluded from its own peer group? Intended yes as of 2026-08-28, in fact type-dependent. `extractAllInstitutions()` now emits `unitId` (line 163) alongside `unitid` and `UNITID`, all three set to `row[unitidIndex]` — the raw cell value. The benchmarker compares `inst.unitId !== institution.unitId && inst.unitId !== institution.unitid`, and `institution.unitId` is set by the wrapper to `institution.unitid`, a String. If the concentration file's UNITID column is numeric, both disjuncts are Number-vs-String and self-exclusion still does not fire, notwithstanding the change block's claim that it now does. UNCLEAR: whether that column is stored as text or as a number in the workbook.

Statistics. Not mean or median of a raw measure — the peer set is used to build a z-score distribution. `__getPeerStats` returns `peerMedianPBI` (true median), `peerTop5Median`, `peerTop5AtCeiling`, `peerTop5Avg` and `peerCSIAvg`. Scoring uses only the **percentile rank** of the institution's PBI within `peerPBIs`; `peerMedianPBI` is reported but never scored. The memo key is `${level}|${peerData.length}|${sorted ids}`.

Certificate PBI (`calculateCertPBI`, lines 522–607) repeats the whole calculation against the certificate file for the applicable cert types, using the same peer set, and is **descriptive only** — `certPbiResults` never enters `totalRiskPoints`.

2026-08 changes

Four blocks: one labelled 2026-08-26, two dated 2026-08-27, one dated 2026-08-28.

1. `worstDirection`, `worstExtremity` and `worstLevel` published (lines 886–904, labelled `PUBLISHED 2026-08-26`). Stated reasoning, quoted: *"These three were computed and then dropped at this boundary, and A3 (Academic Sprawl) and A4 (Monoculture) both gate on worstDirection … `undefined === 'sprawl'` is false, so BOTH archetypes were structurally incapable of firing - not mistuned, unreachable."* Evidence given: *"Confirmed against the v48.4 private C21 run: A3 and A4 fired 0 times in 192 institutions while 27 of them scored a maxed 6/6 on this very analyzer. The signal was there; the classification of it never left this object."* The block adds: *"worstDirection flips on the SIGN of PBI at the driving degree level, so any change to the PBI definition moves the sprawl/monoculture split with it. Publishing the field is also what makes that visible in future runs."* Confirmed downstream: `risk_adjustment_analyzer.js` line 1490 (`checkA3_AcademicSprawl`) and line 1518 (`checkA4_Monoculture`) both read `ar.worstDirection`, and both additionally require `riskPoints >= 6`.

2. Discriminating-level gate (2026-08-27, lines 714–741). Stated reasoning, quoted: *"A degree level may only drive the score if the peer distribution at that level actually spreads."* Evidence given: in the private Carnegie 21 cohort (n=197), *"BA Top5 median 0.487 (real distribution — 197 institutions); MA Top5 median 1.000 (106 institutions, most with <=5 MA programmes); AA Top5 median 1.000 (19 institutions, CSI median 0.000). Scoring off MA or AA there ranked institutions by whether they*

happen to operate a graduate school at all. It put Bard College (70.3 bachelor's degrees per bachelor's programme) at maximum sprawl severity while Hampshire College (2.3) scored zero. Eleven of 51 flags in the 26 Aug private C21 run were MA-driven and two were AA-driven; all thirteen were artefacts of a point-mass distribution." Design note: *"The test is computed from the peer data at run time, not hard-coded, so a level that IS discriminating for another cohort — MA for doctoral universities, AA for community colleges — continues to score normally."* A companion block at line 983 adds the excluded levels to `keyFindings` *"so a suppressed level is visible in the export rather than silently absent"*.

3. Indexing and memoisation (2026-08-27, lines 191–205 and 274–276). Stated reasoning, quoted: *"This analyzer was the last one still doing a linear scan per lookup; the other eight were indexed on 26 Aug. In a ranking run it was the dominant cost: calculatePBI() looped over ~197 peers TWICE, and each iteration called findInstitutionRow() (a full scan of 5,827 concentration rows) plus three findColumnIndex() calls … ~2.3M row comparisons per level, ~7M per institution, ~1.4bn for a 200-institution ranking — quadratic in cohort size, which is why ranking hung while single-institution analysis felt fine."* Three memos added — `__colIdx`, `__unitIndex` (first-wins, *"to match the old scan's semantics"*) and `__peerStats` — with the claim *"no change to any computed value"*, justified for `__peerStats` because *"the peer-derived arrays … depend only on (peerData, level) and NOT on the institution being scored."*

4. `unitId` added to `extractAllInstitutions` (2026-08-28, lines 151–162). Stated reasoning, quoted: *"this method emitted only `unitid` and `UNITID`, but the StandardPeerBenchmarker self-exclusion filter compares `inst.unitId !== institution.unitId`. Both sides of that comparison were undefined, so the filter excluded nobody and every institution was scored against a peer distribution that contained itself — its own Top5/CSI values entered the mean and standard deviation of its own z-scores, dragging those z-scores toward zero."* Evidence given: *"20 institutions are understated by 3 points."* Claimed outcome: *"both the subject … and every peer carry unitId, so self-exclusion actually fires and an institution is benchmarked against peers only."* See **Peer logic** for why that outcome is conditional on the UNITID column's storage type.

Notes

- **The file header (lines 15–20) and the `methodology` string published in every report (lines 929–931) both describe a scoring scheme this code does not implement.** They state `PBI = z(Top5) - z(CSI)`, base points at `|PBI| >= 0.18 -> 6pts, 0.10-0.18 -> 3pts`, and a +2 peer-gap add-on at `>= 0.20`. The code computes `(z(Top5) + z(CSI_adj)) / 2`, scores on percentile extremity at 95/90, and has no add-on. The header and the `methodology` string also invert the direction: they say *"Negative PBI indicates monoculture … positive indicates sprawl"*, while lines 431–433 and line 776 say and implement the opposite.
- The `Moderate` risk level is unreachable.
- `institutionTypeFilter` is passed to the peer benchmarker and silently ignored, so liberal-arts colleges and universities share a peer pool.
- `StandardPeerBenchmarker`'s HBCU filter tests `inst.hbcu === '1'` against the raw cell value from the concentration file; if that column is numeric the `'hbcu-only'` option yields an empty peer group.
- The three "scored 0, level suppressed / no peers / no valid levels" paths are indistinguishable from a genuine 0 in `analysisResults`.
- `calculateStandardDeviation()` (lines 831–834) is dead code — `calculateZScore` computes its own variance inline.
- `static testFrameworkIntegration()` asserts 6 and 6, which matches the framework.

Programs Sustainability

ACADEMIC PROGRAM BREADTH & SUSTAINABILITY

- **Class**: `UnifiedProgramsSustainabilityAnalyzer` (real analyzer, `programs_sustainability_analyzer.js`); `UniversalProgramsSustainabilityAnalyzer` (wrapper, `modular_tool_v48 no data cap.html` lines 1478–1514). Orchestrator key `programssustainability`.
- **Data key**: `'programs_sustainability'`. The real analyzer does **not** call `getDataForAnalyzer`; the wrapper does, at line 1494: `this.dataProcessor.getDataForAnalyzer('programs_sustainability')`, and passes the resulting array to the constructor. Resolved in `data_processing_pipeline.js` line 22 by `/programs.*and.*degrees/i`. Declared `dataRequirements: ['programs and degrees.xls']`.
- **What it measures**: whether the institution's academic portfolio carries enough students per programme, and whether each degree it awards costs more student-facing expenditure than its peers'. The header states: *"Analyzes academic program sustainability through enrollment efficiency and cost metrics"*, over a *"2023 snapshot (no trend analysis)"*. Scoring is entirely relative — every point comes from a quantile or a ratio of the institution's value to its peer group's, never from an absolute level.

Sector differences

There are **no sector-specific thresholds**. `this.sector` is used only to (a) set `sectorName` and (b) filter the peer pool (`peerSector !== this.sector` in both branches of `performPeerAnalysis`). What differs by institution is the **type classification**, which selects which measures are scored. `classifyInstitution()` (lines 143–154) reads `institution.SECTOR || institution.sector` and `institution.C21BASIC || institution.carnegieClassification` — **not** `this.sector` — and walks `this.institutionTypes` in declaration order:

Type	sectorCheck	carnegieCheck (C21BASIC)	riskMeasures
`community_college`	sector `'4'`	always true	`ugrads_per_program`, `sfe_per_degree`
`liberal_arts`	sector `'1'` or `'2'`	`'21'`, `'22'`	`ugrads_per_program`, `sfe_per_degree`
`masters_university`	sector `'1'` or `'2'`	`'18'`, `'19'`, `'20'`	`ugrads_per_program`, `grads_per_program`, `sfe_per_degree`
`research_doctoral`	sector `'1'` or `'2'`	`'15'`, `'16'`, `'17'`	`ugrads_per_program`, `grads_per_program`, `sfe_per_degree`

Fallthrough (line 153) returns `liberal_arts`. Because `community_college` is tested first and its `carnegieCheck` is `() => true`, **every sector-4 institution is a community college regardless of Carnegie class**. Sector `'3'` matches no `sectorCheck` and therefore always lands on the `liberal_arts` default, as does any sector-1/2 institution whose C21BASIC is not one of the eight listed codes.

The practical sector difference is that community colleges and liberal-arts colleges never score `grads_per_program`, which caps their attainable scale points at 5 rather than 6 — see **Maximum points**.

Scoring

Three components plus a bonus. All cut-points are computed at run time from the peer array; there are no fixed numeric thresholds anywhere in this analyzer.

Peer arrays are `peerAnalysis.peerValues[measure]`, filtered to `val !== null && val > 0`, then sorted ascending. Quantiles are positional: `q10 = sorted[Math.floor(n * 0.10)]`, `q25 = sorted[Math.floor(n * 0.25)]`, `q75 = sorted[Math.floor(n * 0.75)]`.

Component 1 — scale risk from `ugrads_per_program` (lines 431–475). Evaluated only if `sustainabilityData.ugrads_per_program !== null`, `peerAnalysis.comparisons.ugrads_per_program` exists, and `peerValues.length > 0`. Branches tested in this order, first match wins:

#	Condition	Floor / ceiling	Points	`trigger`
1	`institutionValue <= q10`	**ceiling** (value ≤ q10)	5	`Q10`
2	`peerGapRatio <= 0.60`, where `peerGapRatio = institutionValue / peerData.average` (null if `peerAvg <= 0`)	**ceiling** (ratio ≤ 0.60)	5	`PeerGap60`
3	`institutionValue <= q25`	**ceiling** (value ≤ q25)	3	`Q25`

#	Condition	Floor / ceiling	Points	trigger
4	otherwise	—	0	null

`scaleRiskPoints += pts`. No cap is applied at this step, so this component alone contributes at most 5.

Component 2 — scale risk from `grads_per_program` (lines 479–525). Gated on `sustainabilityData.grads_per_program !== null`, the peer comparison existing, **and `scaleRiskPoints < 6`**. Same three-branch ladder producing `basePts` of 5 / 5 / 3 / 0. If `basePts > 0` the amount actually added is `Math.min(basePts, 6 - scaleRiskPoints)`, which is what caps the scale block at 6. `measureResults.grads_per_program` is written only when `basePts > 0`, so a grads measure that scores zero is absent from the report rather than present with 0.

Component 3 — cost risk from `sfe_per_degree` (lines 528–559). Single test:

Condition	Floor / ceiling	Points
`institutionValue >= q75`	**floor** (value ≥ q75)	4
otherwise	—	0

`costRiskPoints` is **assigned**, not accumulated, so it is 4 or 0.

Component 4 — combo bonus (lines 562–569).

Condition	Points
`scaleRiskPoints > 0 && costRiskPoints > 0`	2
otherwise	0

`totalRiskPoints = scaleRiskPoints + costRiskPoints + comboBonus`.

Do the components sum to the declared maximum? 6 + 4 + 2 = 12, which equals `this.riskFramework.maxRiskPoints`. But the 6 is only attainable when `grads_per_program` is in `riskMeasures`. Attainable totals are the set {0, 3, 4, 5, 6, 9, 11, 12} for masters/research-doctoral and {0, 3, 4, 5, 9, 11} for community colleges and liberal-arts colleges.

`sfe_per_program` is extracted (line 181) and counted in `sourcesFound`, but is not in any `riskMeasures` array and is scored by nothing — see **2026-08 changes**.

Maximum points

Declared `this.riskFramework.maxRiskPoints = 12` (line 47); the wrapper overwrites `result.analysisResults.maxRiskPoints = 12` unconditionally (line 1508), including on the no-data path. `riskAssessment.maxPossiblePoints` is also hard-coded 12 (line 575). Framework declares 12 in all four sectors.

Reachable for masters and research/doctoral institutions only. 12 requires `scaleRiskPoints === 6`, which requires the `grads_per_program` block to add 1 on top of a 5 from `ugrads_per_program` (or 3+3). `grads_per_program` is absent from the `riskMeasures` array for `community_college` and `liberal_arts`, so for those two types the ceiling is 5 + 4 + 2 = **11 of 12**, and the missing point is structural rather than earned. Every sector-3 institution and every sector-1/2 institution outside the eight listed Carnegie codes also falls to `liberal_arts` and inherits that 11-point ceiling.

There is no clamp inside the analyzer; the orchestrator's `Math.min(Math.max(0, raw), cap)` is the only one, and it never binds because 12 is the maximum the arithmetic can produce.

Risk level mapping

`determineRiskLevel(riskPoints)` (lines 805–812) works on a percentage of the declared maximum, `percentage = (riskPoints / 12) * 100`:

Condition	Level	Equivalent points (out of 12)
`percentage <= 25`	`Low`	0 – 3.0
`percentage <= 45`	`Moderate`	> 3.0 – 5.4
`percentage <= 65`	`High`	> 5.4 – 7.8
otherwise	`Critical`	> 7.8 – 12

Against the attainable set: 0/3 → Low; 4/5 → Moderate; 6 → High; 9/11/12 → Critical. 7 and 8 are not attainable, so the High band is occupied only by the single value 6.

A separate `calculateSustainabilityScore()` publishes `100 - (riskPoints / 12) * 100`, rounded, as `sustainabilityScore`. The report headline is driven by that score (85 / 70 / 55 cut-points) except for one override at lines 866–868: when `scaleRisk >= 3 && costRisk === 0` the headline is replaced with the fixed "maintains peer-equivalent program cost efficiency despite thin per-program scale — viability dependent on cross-deployed faculty" text.

Data-gap behaviour

Path	success	riskPoints	riskLevel	isDataUnavailable	In denominator?
`createNoDataResult` — `sourcesFound === 0` (no institution row, or none of the four columns parsed)	`true`	0	`'Unknown'`	**`true`**	No — excluded
`createErrorResult` — any throw inside `analyze()`	`false`	0	`'Error'`	*not set* (`isError: true` instead)	No — excluded on `!success`
Wrapper early return, `!institution.unitid`	`false`	0	`'Critical'`	*not set*	No — excluded on `!success`. Note the wrapper writes `maxRiskPoints: 8` here, then line 1508 rewrites it to 12
≥1 source found but the peer pool yields no `comparisons` for any measure	`true`	0	`'Low'`	*not set*	**Yes — counted as 0/12**

That last row is the material gap: `hasData` is true as soon as **one** of the four columns parses, so an institution with a `Ugrads per program` value but no usable peer group scores a clean 0 of 12 and is labelled Low, with nothing in `analysisResults` distinguishing it from an institution that genuinely scored zero.

No path awards points for missing data. Every failure path returns 0.

Peer logic

`performPeerAnalysis()` (lines 258–404) has two branches, chosen on whether `peerCriteria.carnegieBasic` is non-empty. `StandardPeerBenchmarker` is **not** used by this analyzer even though `hasEnhancedPeerFiltering` is computed (line 42) and logged; the flag is never read again.

Branch A — `peerCriteria.carnegieBasic` present and non-empty (lines 262–289): filters `this.institutions` (built from the programs-and-degrees file itself) by, in order: self-exclusion `peer.unitid === institution.unitid || peer.UNITID === institution.UNITID`; `peerSector !== this.sector`; Carnegie match against `peer.carnegieBasic` or `peer.C21BASIC`; `peerCriteria.hbcuFilter`; `peerCriteria.customPeerGroup`.

Branch B — otherwise (lines 292–316): self-exclusion, sector filter, then `classifyInstitution(peer)` must return the same institution type; then the same HBCU and custom-peer-group filters.

Is the institution excluded from its own peer group? Intended yes, in fact type-dependent and probably not. `extractAllInstitutions()` line 688 stores `unitId = row[unitidIndex]` — the raw cell value, un-normalised — into both `unitid` and `UNITID`. The subject's `institution.unitid` is a String (pipeline line 241). The comparison is `===`. If the UNITID column in `programs and degrees.xls` is numeric, both disjuncts are Number-vs-String and the institution stays in its own peer pool, entering its own `average`, `median` and q10/q25/q75. UNCLEAR: whether that column is stored as text or as a number in the workbook.

Statistics. For each measure in `typeConfig.riskMeasures`, `peerComparisons[measure]` carries `average` (arithmetic mean), `count`, `median` (true median — `calculateMedian()` averages the two middle values for even n), `min`, `max` and the raw `values` array. `peerValueArrays[measure]` carries the same raw array and is returned as `peerAnalysis.peerValues`, which is what the quantile cut-points read. Only `average` and the quantiles are used for scoring; `median`, `min` and `max` are reported only.

Certificate-efficiency peer comparisons (`ug_enroll_per_cert_program`, `gr_enroll_per_cert_program`, lines 362–393) are computed over the same peer set and are descriptive only.

`calculatePeerAverageForMeasure()` (lines 746–795), used for the descriptive programme and degree counts, is a separate and much looser pool: it filters `this.institutions` by institution type alone (lines 757–763) — no sector filter, no

self-exclusion, and none of the Carnegie, HBCU or custom-peer-group criteria. This is the same defect the 2026-08-28 change block describes for the now-deleted `getPeerValuesForMeasure`, still live for the descriptive statistics.

2026-08 changes

Three dated blocks, all `2026-08-28 (v49.1)`, plus one `2026-08-27` performance note.

1. `sfe_per_program` removed from every `riskMeasures` array (lines 51–61). Stated reasoning, quoted: *"WAS WRONG: the measure was listed for all four institution types, was peer-benchmarked in performPeerAnalysis, and was named in the generated methodology string (which is built from riskMeasures.join), but calculateRiskAssessment contains no branch that scores it — it contributed 0 of the 12 points while every report claimed it had been evaluated."* Evidence given: *"calculateRiskAssessment scores only ugrads_per_program, grads_per_program and sfe_per_degree (plus the scale+cost combo bonus); no code path reads peerAnalysis.comparisons.sfe_per_program."* Outcome: *"the measure is no longer advertised or benchmarked. Scores are unchanged (it never scored); scoring for it is deliberately NOT added here."* The measure is still extracted at line 181 and still counted in `sourcesFound`.

2. Raw per-measure peer value arrays retained and returned (lines 320–333 and 399–401), and **3. `getPeerValuesForMeasure` deleted** (lines 607–616). These are one change. Stated reasoning, quoted: *"the q10/q25/q75 cut-points that award every point in this measure were re-derived by getPeerValuesForMeasure, which rebuilt the pool from institution type alone — no sector filter, no self-exclusion, and none of the Carnegie / HBCU / custom-peer-group criteria applied here — so the cut-points came from a different and much larger population than the peer averages printed beside them."* Evidence given: *"public liberal arts q10 computed as 16.6 ugrads/programme against 39.7 sector-correct, q25 28.5 against 61.9; private research/doctoral q10 48.3 against 37.2 sector-correct. 138 of 1,536 institutions change score."* Outcome: *"quantiles and averages are drawn from one and the same filtered, self-excluded peer set collected in this loop"*, read by callers as `peerAnalysis.peerValues[measure]`. The three scoring blocks carry matching one-line notes at lines 434, 482 and 531.

4. `findInstitutionRow` indexed (2026-08-27, lines 639–644). Stated reasoning, quoted: *"This was called once per peer inside three separate peer loops (ugrads_per_program, grads_per_program, sfe_per_degree), each call scanning all 5,828 rows of 'programs and degrees'. ~197 peers x 3 loops x 5,828 rows = 3.4M row comparisons per institution — quadratic in cohort size for a ranking run. First-wins semantics preserved; keys are String()+trim() so the original loose == comparison still holds."* The `Map` is built with `String(v).trim()` keys and queried with `String(unitid).trim()`, so this lookup — unlike the peer self-exclusion above — is type-safe.

Notes

- 12 of 12 is unreachable for community colleges and liberal-arts colleges, and for every institution that falls through `classifyInstitution` to the `liberal_arts` default.
- The self-exclusion in both `performPeerAnalysis` branches uses `===` on values of possibly different types; see **Peer logic**.
- `calculatePeerAverageForMeasure` still uses a type-only, self-inclusive, sector-blind pool.
- `extractAllInstitutions()` (lines 674–708) never sets an `hbcu` field, so `peerCriteria.hbcuFilter === 'hbcu-only'` filters the peer group to empty and `'non-hbcu-only'` filters nothing.
- `calculateMeasureRisk()` (lines 585–605) is dead code — a ratio-band scorer, never called.
- `dataContext` reports *"${sourcesFound} of 4 key measures available"*, where the 4 still includes the unscored `sfe_per_program`.
- `static testFrameworkIntegration()` (lines 990–1003) asserts `publicPoints === 8 && privatePoints === 8`; the framework declares 12, so the test returns false whenever `RiskArchitectureEngine` is defined.
- Quantiles are positional (`sorted[Math.floor(n * p)]`) with no interpolation, so q10 and q25 coincide for small peer groups; there is no minimum peer count guarding the quantile path (only `peerValues.length > 0`).

Research Support

OPERATING PERFORMANCE & COST STRUCTURE

- **Class**: `UnifiedResearchGrantsAnalyzer` (real analyzer); `UniversalResearchGrantsAnalyzer` (wrapper, lines 1811–1867). Orchestrator key `researchgrants`.
- **Data key**: `'research_grants_expenses'` (wrapper line 1829). Resolved in `data_processing_pipeline.js` line 43 by `/new.*annualized.*research.*grants.*exps/i`. Columns read are `'{year} ggc'` and `'{year} rexp'` for 2010–2024.
- **What it measures**: *"Exposure to declining federal research support. Federal research revenue falling behind what comparable institutions are achieving is the risk."* Two peer-relative CAGR comparisons carry the whole measure: government grants and contracts growth, and coverage growth (coverage = government grants and contracts ÷ total research expenditure).

The header is equally explicit about what is deliberately **not** scored:

> *"It does not score the LEVEL of coverage. A low coverage ratio is ambiguous on the current data: it may mean the institution is subsidising research from its own funds (risk), or that it has already diversified away from federal money (less risk). IPEDS does not let us tell those apart. Until it does, the level is reported and not scored. Revisit for the 2025 load if a way to observe non-federal research funding appears. It also does not score absolute grant volume against peers. An institution is not at risk for winning less federal money than a larger one; it is at risk when its own federal support is going backwards relative to its peer group."*

Sector differences

None in the scoring. `this.riskFramework.maxRiskPoints = 8` and both component maxima of 4 are hard-coded with no sector branch. The header states *"Max Risk Points: 8 (both sectors)"*, and the framework note at `risk_scoring_engine.js` line 82 confirms 8/8 as intended for public and private alike.

The applicability gate is **Carnegie, not sector**: `SCORED_CARNEGIE = [15, 16]`. Anything else returns `createNotApplicableResult()`. The narrowing is explained:

> *"Population narrowed from Carnegie 15/16/17 to 15/16. C17 is doctoral or professional with little or no research operation: its median reported coverage was 160% and half the group exceeded 150%, i.e. a research expenditure denominator too small to form a ratio from. C15 runs 24-193%, median 74%."*

`risk_scoring_engine.js` sets **0 points for community college** (*"EXCLUDED - CCs don't have research missions"*), which triggers the orchestrator's framework-zero path: points forced to 0 and `riskLevel` rewritten to `'Informational'`.

`this.sectorName` is `selectedSector === '1' ? 'Public' : 'Private'`, so sector `'4'` is labelled "Private" in log lines.

Scoring

`scoreCAGRComponent()` (lines 496–502). `gap = institutionCAGR − peerCAGR`, both in percent-per-year, so the gap is in **percentage points per year**. **Both thresholds are ceilings: the tier fires when `gap <= X`.**

Condition	points (each component, maxPoints = 4)
gap <= −3 pp/yr (3+ pp/yr behind peers)	4
gap <= −1 pp/yr (1–3 pp/yr behind peers)	2 (= maxPoints / 2)
gap > −1 (keeping pace or ahead)	0

Component 1 — Government Grants Trend (lines 442–458), max 4. Institution `ggcCAGR` against the peer group's **mean** `ggcCAGR`. Scored only when both are non-null.

Component 2 — Coverage Trend / "Gov Grants as % of Research" (lines 460–477), max 4. Institution `ratioCAGR` against the peer group's **mean** `ratioCAGR`. Scored only when both are non-null.

CAGR is computed by `calculateTrends()` (lines 407–435) between the **first and last usable observations** (`series[i] !== null && series[i] > 0`), over the actual span between them: `(Math.pow(last / first, 1 / span) - 1) * 100`. Returns `cagr: null` when fewer than two usable points exist, or when span, first or last is not positive.

Coverage series (lines 370–374): `ggc / rexp`, `null` wherever `rexp === null || rexp <= 0 || ggc === null || ggc < 0`.

Do the components sum to the declared maximum? Yes: 4 + 4 = **8**. Reweighted in this change from *"3+2 -> 4+4"* after a third component was removed.

Because each component returns only `maxPoints`, `maxPoints / 2` or `0`, the **only attainable totals are 0, 2, 4, 6 and 8**.

Maximum points

`this.riskFramework.maxRiskPoints = 8`, published in every result path. The wrapper re-asserts 8 on the success path (line 1857).

Reachable? Yes — both components at `gap <= -3` give 8. But only when **both** peer means are non-null; if either is null its component is silently skipped and the reachable total halves to 4 while the published denominator stays 8. See Defects.

Risk level mapping

`determineRiskLevel()` (lines 556–561), on raw points:

score	riskLevel
<= 0	`Low`
<= 2	`Moderate`
<= 4	`High`
> 4	`Critical`

Against the five attainable totals: 0 → `Low`, 2 → `Moderate`, 4 → `High`, 6 and 8 → `Critical`.

The 2026-08-28 note records why this was changed from percentages: *"With two 4-point components the only attainable scores are 0, 2, 4, 6 and 8; the previous percentage bands (<=25 Low, <=45 Moderate, <=65 High, else Critical) made 'Moderate' unreachable, since it required 2.08-3.6 points."*

Data-gap behaviour

Four terminal paths in the analyzer, with **different** flags:

Path	success	riskLevel	isDataUnavailable	riskPoints	maxRiskPoints
`createNotFoundResult` — unitid absent from the research file	`false`	`not_applicable`	**not set**	0	8
`createNotApplicableResult` — Carnegie outside [15, 16]	`true`	`not_applicable`	**not set**	0	8
`createInsufficientDataResult` — < 2 usable years, or either scored CAGR null	`false`	`insufficient_data`	**true**	0	8
`createErrorResult` — thrown error	`false`	`error`	**not set**	0	8

The two `not_applicable` paths are excluded from the orchestrator's denominator by the `naLevel` test, and `insufficient_data` by the `isDataUnavailable` test. The `error` path is excluded by neither — see Defects.

The 2026-08-28 guard (lines 196–205) is explicit: *"Both scored trends must be formable. A null CAGR means the underlying series had no usable first or last year, which is a data gap, not a finding. Scoring it would hand maximum risk to an institution whose 2024 cell happens to be blank."*

Wrapper-level gaps (lines 1817–1846). Both award points and neither sets `isDataUnavailable`:

- institution object missing `unitid` → `{ riskPoints: 22, riskLevel: 'Critical' }`
- `research_grants_expenses` dataset absent → `{ riskPoints: 22, riskLevel: 'Critical' }`

Peer logic

Two paths, chosen by whether `StandardPeerBenchmarker` is defined at construction time (line 93).

Path A — `StandardPeerBenchmarker` (`performStandardizedPeerAnalysis`, lines 250–272). `benchmarker.findAndAnalyzePeerGroup({unitId: 'dummy'}, peerCriteria)`. **Self-exclusion: NO** — the benchmarker's self-exclusion step (lines 39–41 of `standard_peer_benchmarker.js`) compares against the literal `'dummy'`, so the institution remains in its own peer group. This is acknowledged in the header as known and unchanged:

"performStandardizedPeerAnalysis passes {unitId: 'dummy'} to the benchmarker, so an institution is not excluded from its own peer group." The benchmarker applies, in order: self-exclusion (ineffective), Carnegie array filter, HBCU filter, custom peer group, then `inst.sector === this.sector`. **When `peerCriteria.carnegieBasic` is absent, no Carnegie narrowing is applied at all on this path** — the `SCORED_CARNEGIE` default exists only in Path B.

Path B — legacy (`performLegacyPeerAnalysis`, lines 275–319), used when `StandardPeerBenchmarker` is undefined or its call throws. **Self-exclusion: yes** (line 280). Sector filter, then Carnegie: if `peerCriteria.carnegieBasic` is supplied it is used, **otherwise** the population is narrowed to `SCORED_CARNEGIE` = [15, 16]. Then HBCU and custom peer group filters.

Statistics. `calculatePeerTrends()` (lines 324–347) returns arithmetic **means** of each peer's `ggcCAGR`, `rexpCAGR` and `ratioCAGR`, over peers whose CAGR could be formed. `peerCount` is `Math.max` of the three array lengths. This is flagged in the header as a known limitation:

> *"Peer benchmarks are MEANS, not medians, so a single fast-growing peer lifts the bar for the whole group (C16 privates are compared against +4.14%/yr, with one peer at +33%/yr in that pool). Left as-is to keep this change to one thing; a median is the better statistic here."*

`calculatePeer2024Averages()` (lines 629–660) computes separate **means** of last-available government grants, research expenditure and coverage, used only in key findings; it returns `0` for each when no peers qualify.

`rexpCAGR` is computed and averaged but **not scored** — the component that used it was removed.

2026-08 changes

All changes in this file are dated **2026-08-28** and stem from the redesign block at lines 9–71 ("Specified by D. Greenstein").

Sign correction (lines 488–495, the central fix). *"scoreCAGRComponent SIGN CORRECTED. It previously awarded FULL points for matching or beating peer CAGR and ZERO for falling well behind, so the measure ran backwards. In the 2026 baseline this put MIT, Johns Hopkins, Yale and Columbia at 8/8 'Critical' while institutions whose grant coverage had collapsed to 0% scored 0/8 'Low'."* The prior implementation is quoted verbatim in the code:

```
if (gap >= -1) return maxPoints;              // matching peers = FULL points
if (gap >= -3) return Math.floor(maxPoints/2);
return 0;                                     // far behind peers = ZERO
```

Component removed. *"Component 'Research Expenditures Trend' (3 pts) REMOVED. It scored an institution for spending more on research than its peers, with no reference to whether funding kept pace. Not a risk quantity on its own."*

Reweighting. *"Remaining two components reweighted 3+2 -> 4+4."*

Population narrowed (lines 82, 291, 539) — Carnegie 15/16/17 → 15/16, with the C17 coverage evidence quoted above.

Missing years are null, not 0 (line 350). *"Previously a blank cell became 0, so a missing 2010 value produced a CAGR computed off a denominator of 1 (a $50M 2024 value yielded roughly +328%/yr), and a missing 2024 value produced exactly -100%/yr. Those artefacts scored as safe under the old inverted logic and would score as maximum risk under the corrected one."*

CAGR rebuilt (line 402). *"returns cagr null rather than fabricating a value. Previously `const cagr = Math.pow(last / (first || 1), 1/(series.length-1)) - 1;` substituted a denominator of 1 for a missing first year and returned exactly -100% for a missing last year. It also always divided by 14, regardless of how many years actually separated the two endpoints."*

Peer CAGR averages cleaned (line 322). *"peers whose CAGR cannot be formed are now excluded from the average rather than contributing a -100% or a denominator-of-1 artefact."*

Data-gap guard added (line 196), quoted above.

Risk level mapped from points (line 552), quoted above.

Notes

- `setInstitutions()` (lines 115–123) accepts the system institution list and discards it; the analyzer always uses its own internal extraction.
- `extractAllInstitutions()` (lines 151–158) builds peer objects with `unitId`, `name`, `carnegie`, `sector`, `carnegieBasic` and `rowData` — **no `hbcu` property**. See Defects.
- `findColumnIndex()` here is an **exact, case-insensitive equality** match, unlike the substring matcher in the enrolment and UNA analyzers.

- `formatCurrency()` (lines 662–668) returns `'$0'` for zero/null but returns `'1.2M'`, `'500K'`, `'750'` — **without a $** — for every other value.
- Key findings and the narrative label the CAGR span *"2010-2024"* unconditionally, though `calculateTrends` uses the first and last usable years and reports the real `span` internally.

Return on Net Assets

LIQUIDITY & BALANCE SHEET

- **Class**: `UnifiedReturnOnNetAssetsAnalyzer`
- **Data key**: `'return_on_net_assets'` (wrapper `UniversalReturnOnNetAssetsAnalyzer`, HTML line 1250). Internal `this.measureName = 'return_on_net_assets'`; `dataRequirements = ['annualized return on net assets.xls']`. Wrapper label string is `'Return on Net Assets Analysis'`.
- **What it measures**: Header comment: "Measures change in net assets relative to total net assets - indicates operational efficiency. Positive values = generating returns, negative values = consuming assets. Higher values = stronger operational performance and resource management." Units are declared as `'percentage'`; values are read from year columns as-is (no rescaling).
- **Sector differences**: **None in scoring.** All five factors use the same absolute thresholds for every sector — this analyzer has no percentile-relative public branch and no restriction-adjusted path. Sector affects only (a) the `sectorName` display string (`'1'` Public, `'2'` Private, `'3'` For-Profit, else Community College) and (b) the composition of the peer group, since `extractAllInstitutions` drops rows whose sector does not match `this.sector`.
- **Scoring**:

`maxPoints = measureConfig.maxRiskPoints = 10`. Status for Factors 1 and 2 comes from `assessRONAStatus`, using `measureConfig.riskThresholds` (`critical: -5.0, moderate: 0.0, adequate: 3.0, excellent: 3.0`):

Test (ceiling)	Status
value < -5.0	critical
value < 0.0	concerning
value < 3.0	adequate
value >= 3.0	excellent
null / undefined	unknown

Factor 1 — current value (2 points, "20% weight"). Runs if `currentValue !== null`.

Status	Points
critical	2
concerning	1.5
adequate	0.5
excellent	0 (no branch, no factor text)

Factor 2 — 3-year rolling average (4 points, "40% weight"). Runs if `rollingAverage !== null`.

Status	Points
critical	4
concerning	3
adequate	1
excellent	0

Factor 3 — trend direction (2 points, "20% weight").

`measureAnalysis.trend`	Points
`'declining'`	2
`'stable'`	0.5
`'improving'`	0
`'insufficient-data'`	0

Trend uses **absolute** change over the full available span, not percent — "For RONA, use absolute change rather than percent change since values can be negative": `> 2.0` improving, `< -2.0` declining, else stable.

Factor 4 — volatility (1 point, "10% weight"). Population standard deviation of all year values:

Volatility (floor, exclusive)	Points
> 5.0	1
> 3.0 (and <= 5.0)	0.5
<= 3.0	0

Factor 5 — peer comparison modifier (1 point, "10% weight"). Requires `peerAnalysis.success` and a non-null `currentPerformance.percentile`:

Percentile (ceiling)	Points
< 25	1
< 50	0.5
>= 50	0

Sum is capped: `Math.min(riskPoints, 10)`.

- **Maximum points**: `measureConfig.maxRiskPoints = 10`, returned as `maxPoints`. The HTML wrapper forces `maxRiskPoints = 10` on success. Factor caps 2 + 4 + 2 + 1 + 1 sum exactly to 10.
- **Risk level mapping**: from `riskPercentage = (riskPoints / 10) * 100`, computed before the `Math.min` cap:

Condition (floor)	riskLevel
riskPercentage >= 65	Critical
riskPercentage >= 45	High
riskPercentage >= 25	Moderate
otherwise	Low

- **Data-gap behaviour**: `createFallbackResult` (no data, UNITID column missing, institution not found, "Insufficient return on net assets data") returns `success: false`, `analysisResults: { currentValue: null, riskPoints: 0, maxRiskPoints: 10, riskLevel: 'Unknown', isDataUnavailable: true }` and an empty `peerGroup`. `createErrorResult` (missing unitId, thrown exception) returns `riskLevel: 'Unknown'`, `isDataUnavailable: true`, `riskPoints: 0`. The HTML wrapper's missing-`unitid` return is `{ riskPoints: 0, maxRiskPoints: 10, riskLevel: 'Unknown' }` with **no** `isDataUnavailable` flag.
- **Peer logic**: Identical structure to the capitalization ratio analyzer — `StandardPeerBenchmarker` when the global is defined, otherwise `basicPeerFiltering`. Self-exclusion, Carnegie basic, HBCU, `customPeerGroup`, then `inst.sector === this.sector`; the candidate pool is already sector-filtered in `extractAllInstitutions`. The scoring comparison is a **percentile rank** from this analyzer's own `benchmarkValue` (peers strictly below ÷ peer count, higher-is-better), mapped `>= 75` Top quartile/Excellent, `>= 50` Above median/Good, `>= 25` Below median/Fair, else Bottom quartile/Concerning. Report findings quote the peer **median**; `generatePeerComparison` reports position against both mean and median.
- **Notes**:
  - The `excellent` and `adequate` thresholds are both `3.0`, so the `adequate` band is `0.0 <= value < 3.0` and there is no gap; unlike the capitalization analyzer, no band here is unreachable.
  - No `__unitRow` UNITID index was added to this file. `extractInstitutionData` still performs a linear scan of every data row per lookup — the pattern the 2026-08-26 comments in the viability, primary reserve and capitalization analyzers describe as the cause of ranking-run failures. There is no dated comment in this file at all.
  - `measureConfig.riskWeights` (0.2 / 0.4 / 0.2 / 0.1) is declared but never read.
  - `analyzePeerMeasureData` guards peer values with `!== null`; `recent3Year` is explicitly `null` when fewer than three years exist, so that guard holds here.
  - Rolling 3-year averages are consecutive triples over the *available* year list; calendar gaps are bridged silently. `recent3Year` is the mean of the last three available values.
  - The constructor's `console.log` is unconditional here — it lacks the `window.__RAT_QUIET` suppression present in the viability, primary reserve and capitalization analyzers.

Salaries & Benefits

OPERATING PERFORMANCE & COST STRUCTURE

- **Class**: `UnifiedSalariesBenefitsAnalyzer` (wrapper: `UniversalSalariesBenefitsAnalyzer`, wrapper label `'Salaries and Benefits Analysis'`, analyzer key `salariesbenefits`)
- **Data key**: The analyzer does not call `getDataForAnalyzer` itself. The wrapper (line 2237) calls `this.dataProcessor.getDataForAnalyzer('salaries_benefits')`, which maps to `/annualized.*salaries.*benefits.*FTE.*pv.*(and|&).*pu/i`.
- **What it measures**: Salaries and benefits **per FTE**, read from columns named `{year}per` for 2010–2024, scored on three things: the compound annual growth rate relative to the peer mean growth rate, the current per-FTE level relative to the peer median, and the year-over-year volatility of the per-FTE series. Total salaries (columns named as a bare `{year}`) are extracted and reported but **not scored**.
- **Sector differences**: **None in the thresholds or the maximum.** `riskConfig.maxPoints = 6` unconditionally, and all three threshold groups are sector-independent. `this.sector` affects only (a) `dataRequirements`, which names `'annualized salaries and benefits with per FTE pu.xls'` for sector '1' and the pv file for every other sector, and (b) the peer pool, which `extractAllInstitutions` restricts to rows whose SECTOR string equals the caller's. The wrapper does **not** override `maxRiskPoints`. `risk_scoring_engine.js` declares 6 points in the public, private, for-profit and community-college blocks alike.
- **Scoring**:

Factor 1 — per-FTE growth vs peer mean growth (0–3). `growthDifference = institutionGrowth - peerAverageGrowth`, both being compound annual growth rates in percent. Thresholds are **floors** (>=).

Condition	Points
`growthDifference >= 15`	3
`growthDifference >= 10`	2
`growthDifference >= 5`	1
`growthDifference > 0`	0
`growthDifference <= 0`	0

Factor 1 **fallback** — used when no peer average growth exists. Scored on the institution's own growth rate. Thresholds are **strict floors** (>).

Condition	Points
`institutionGrowth > 15`	3
`institutionGrowth > 8`	2
`institutionGrowth > 5`	1
`institutionGrowth <= 5`	0

Factor 2 — per-FTE cost vs peer **median** (0–2). `percentDifference = ((institutionPerFTE - peerMedian) / peerMedian) * 100`. Thresholds are **floors** (>=).

Condition	Points
`percentDifference >= 25`	2
`percentDifference >= 15`	1
`percentDifference >= 5`	**0**
`percentDifference >= 0`	0
`percentDifference < 0`	0

Factor 3 — volatility of the per-FTE series (0–1). `volatility` is the root-mean-square of the year-over-year percent changes. Thresholds are **floors** (>=).

Condition	Points
`volatility >= 15`	1

Condition	Points
`volatility >= 10`	**0**
`volatility < 10`	0

The three components sum to 3 + 2 + 1 = 6, which **is** the declared maximum, so `Math.min(totalRiskPoints, 6)` is a no-op.

- **Maximum points**: `maxRiskPoints = 6`, hard-coded as a local in `assessSalaryRisks` and duplicated in `riskConfig.maxPoints`. **Reachable** — requires `growthDifference >= 15` (or `institutionGrowth > 15` on the fallback), `percentDifference >= 25` and `volatility >= 15` together.
- **Risk level mapping**: computed in `assessSalaryRisks` and published directly as `analysisResults.riskLevel`:

Condition	riskLevel
`totalRiskPoints >= 5`	`High`
`totalRiskPoints >= 3`	`Moderate`
otherwise	`Low`

There is no `Critical` band.

- **Data-gap behaviour**: `createErrorResult` (institution row not found → `DATA_NOT_FOUND`; no per-FTE current value → `NO_SALARY_DATA`; thrown error → `ANALYSIS_FAILED`) returns `riskPoints: 0`, `maxRiskPoints: 6`, `riskLevel: 'Unknown'`, `isDataUnavailable: true`. Within the scorer, a missing growth series adds 0 with the factor string `'Salary growth trend unavailable - not scored'`, a missing peer median adds 0 with `'Peer cost comparison unavailable - not scored'`, and a missing volatility skips Factor 3 silently. **No inner path awards points for missing data** (both former +1 charges were removed on 2026-08-29). The wrapper's missing-institution early return publishes `riskPoints: 3, riskLevel: 'Moderate'` with no `isDataUnavailable` — half the measure's points awarded for an absent institution object.
- **Peer logic**: `findAndAnalyzePeerGroup` filters `this.institutions` (already sector-restricted) with a self-exclusion matching either `institution.unitid` or `institution.unitId`, then a Carnegie filter reading `peerCriteria.carnegieFilter ?? peerCriteria.carnegieBasic` (normalised to an array — see the 2026-08-29 change), `peerCriteria.hbcuFilter` and `peerCriteria.customPeerGroup`. Each peer's row is then read and only peers with a truthy `perFTE.current` contribute to the statistics. **Two different statistics are used**: Factor 1 compares against the peer arithmetic **mean** growth rate (`averages.perFTEGrowthRate`); Factor 2 compares against the peer **median** per-FTE cost (`analysis.perFTE.median`). `calculatePercentileRanking` and `calculatePercentile` produce quartile/percentile strings for the report only. The institution **is** excluded from its own peer group. Note that `peerAnalysis.count` is the count of peers **before** the data check, while `analysis.perFTE.count` is the count with usable data; the published `peerGroup.count` uses the former.
- **2026-08 changes**: four blocks, all dated 2026-08-29 (v49.1). 1. `findAndAnalyzePeerGroup` Carnegie filter (line 124) — **array handling**: *"this compared `carnegieBasic.toString()` against the institution's classification as a single string. Every other analyzer passes carnegieBasic as an ARRAY. A one-element array survived by accident - [16].toString() is '16' - but a multi-Carnegie selection produced '15,16', matched nothing, and returned a peer group of ZERO with no error. The measure still scored, on its absolute fallback, and nothing in the output said the peer comparison had not happened."* 2. Factor 1 data gap (line 472): *"was +1. A missing growth series was charged as risk. Data gaps never score."* 3. Factor 2 ladder (line 492) — **rebalanced from 3/2/1 to 2/1/0**: *"this ladder awarded 3/2/1 while its own header declares Factor 2 as '0-2 points'. The three factors summed to 7 against an enforced maximum of 6, so Math.min silently truncated any institution that scored on all three - and which point was lost depended on nothing meaningful. Now 2/1/0, so growth (3) + cost (2) + volatility (1) = exactly 6."* 4. Factor 2 data gap (line 514): *"was +1. An absent peer comparison was charged as risk. Data gaps never score."*
- **Notes**:
  - The 2026-08-29 Factor 2 rebalance leaves the `costThresholds.moderate` band (5–15% above peer median) awarding **0 points** — a live threshold that awards nothing, with an emitted factor string (`'Moderately higher costs: …'`) that still reads as a finding. The same is true of `volatilityThresholds.moderate` (10–15%), which also awards 0.
  - `findInstitutionRow` matches on **`row[0]`**, hard-coding UNITID to the first column, while `extractAllInstitutions` locates the UNITID column by searching the headers for `'unitid'`. If UNITID is not column 0 in the salaries file, `findInstitutionRow` fails for every institution and every peer while

institution extraction still succeeds. **UNCLEAR: whether UNITID is column 0 in the delivered salaries file** — the file is not present in the working directory.

- Factor 2's gate is `if (institutionPerFTE && peerAnalysis?.analysis?.perFTE?.median)`, a truthiness test: a per-FTE value or peer median of exactly 0 routes to the not-scored branch.
- `calculateSalaryTrends` returns `null` when `yearSpan === 0` or `firstValue <= 0`, so a single-year or zero-baseline series produces no growth rate and no volatility, and Factors 1 and 3 are both skipped.
- `analyze()` requires `salaryAnalysis.perFTE?.current` to be truthy; an institution with total-salary data but no per-FTE columns is treated as `NO_SALARY_DATA`.
- `assessSalaryRisks`' emitted factor strings for the `growthDifference > 0` and `growthDifference <= 0` bands both award 0 but describe opposite situations; only the text differs.

State Appropriations Dependency

REVENUE DEPENDENCE & PRICING

- **Class**: `SimpleAppropriationsAnalyzer` (real analyzer); `UniversalAppropriationsAnalyzer` (wrapper, lines 2294–2352). Orchestrator key `appropriations`.
- **Data key**: `'appropriations'`, with `'annualized_total_pub_approps_pu'` as a `??` fallback (wrapper lines 2332–2333). Resolved in `data_processing_pipeline.js` line 49 by `/annualized.*total.*pub.*approps/i`.
- **What it measures**: exposure to a single revenue source — state appropriations as a percentage of core revenues, for public institutions and community colleges. Higher dependency is treated as higher political and budget risk. The header states it is the mirror of the tuition dependency measure: *"The two were designed as mirror images and were not built as such; they now share one shape — same three factors, same band form, same peer method, same vocabulary."*

Sector differences

`isPublicLike = (sector === '1' || sector === '4')`. For any other sector `maxPoints` is **0**, all band arrays are empty, and both peer awards are 0. `analyze()` returns `createNotApplicableResult()` immediately for sectors other than `'1'` and `'4'` (lines 94–96), and the constructor logs a warning.

The wrapper adds a second, earlier exit for **sector `'2'` only** (lines 2301–2318): `{ riskPoints: 0, maxRiskPoints: 0, riskLevel: 'not_applicable' }`. Sector `'3'` falls through to the analyzer's own not-applicable result.

Public and community college are scored identically — there is no band difference between them anywhere in the file.

Scoring

`scoreBand()` (lines 246–250) is `for (const b of bands) { if (value >= b[0]) return b[1]; }` — **level and trend thresholds are floors: the band fires when `value >= X`.** `valueScale: 100` records that *"Series is already expressed in percent (0-100)"*; no conversion is applied.

Factor 1 — absolute dependency level, in percent (`levelBands`, line 73):

Condition	public / CC	private, for-profit
dependency >= 70%	9	n/a (no bands)
dependency >= 50%	7	n/a
dependency >= 30%	5	n/a
dependency >= 20%	3	n/a
below 20%	0	n/a

Factor 2 — trend, change in dependency in PERCENTAGE POINTS over the span (`trendBands`, line 76). Span is first to last usable year in 2010–2024. *"A falling or flat dependency scores zero; only a rise scores."*

Condition	public / CC
rise >= +10 pp	2
rise >= +5 pp	1
otherwise	0

Factor 3 — peer position, percentile rank (`peerBands` line 79, applied in `calculatePeerRisk()` lines 269–290). *"LOWER dependency is better, so the percentile counts peers with a HIGHER ratio, and the bottom quartile is the worst quartile."*

Quartile (percentile band)	public / CC
`Bottom quartile` (percentile < 25 — highest dependency)	4
`Below median` (25 <= percentile < 50)	2
`Above median` (50 <= percentile < 75)	0
`Top quartile` (percentile >= 75)	0
`N/A` (no peer values, or dependency null)	0

Do the components sum to the declared maximum? Yes: 9 + 2 + 4 = **15**. The header records both the weighting and what moved: *"WEIGHTING (public / CC, 15 points): level 9, trend 2, peer 4… The level cut-points (20/30/50/70) and the intermediate awards (3, 5, 7) are unchanged. Only the top award moves, 10 -> 9, to make room for peer."*

Maximum points

`this.riskConfig.maxPoints` — **15 for public and community college, 0 otherwise.**

Reachable? Yes. Dependency >= 70%, a rise of >= 10 pp, and a bottom-quartile percentile are mutually compatible. The header notes the previous state: *"dependency.maxPoints was 15 while only 10 was reachable."*

Risk level mapping

`calculateTotalRisk()` (lines 317–326) produces **two** fields from the same `percentage = totalPoints / maxPoints × 100`.

`riskLevel` — the value the framework, report and layer 2 read:

percentage	riskLevel	in points (of 15)
>= 80	`Critical`	12–15
>= 55 (and < 80)	`High`	9–11
>= 30 (and < 55)	`Moderate`	5–8
otherwise	`Low`	0–4

(8 points is 53.3%, below the 55 cut, so it falls to `Moderate`; 9 points is 60%.)

`dependencyBand` — retained for the narrative text only, on different cut-points and a lower-case five-value vocabulary:

percentage	dependencyBand
> 75	`critical`
> 50	`high`
> 25	`moderate`
> 10	`limited`
<= 10	`minimal`

`dependencyBand` — not `riskLevel` — drives the recommendations list, the headline, the narrative and `generateStrategicImplications()`.

Data-gap behaviour

- **No appropriations rows, institution not found, or no usable year value** (`extractAppropriationsData` returns `hasData: false`) → `createErrorResult()` (lines 685–703): `success: false`, `riskPoints: 0`, `riskLevel: 'Unknown'`, **`isDataUnavailable: true`**, `maxRiskPoints: this.riskConfig.maxPoints`, `currentValue: 0`.
- **Wrong sector** → `createNotApplicableResult()` (lines 666–683): `success: true`, `riskPoints: 0`, `riskLevel: 'not_applicable'` (lower-case, non-house token), **`isDataUnavailable: true`**, `maxRiskPoints: 0`.
- **Any thrown error** inside `analyze()` → the same `createErrorResult()`.
- **Wrapper-level gaps**: sector `'2'` early return (0/0, `'not_applicable'`, **no** `isDataUnavailable`); institution object missing `unitid` → `{ riskPoints: 0, riskLevel: 'High' }` with **no** `isDataUnavailable` and **no** `maxRiskPoints`.
- **No path awards points for missing data.** Null level and null trend both score 0 via `scoreBand`'s `isNaN`/null guard; a null dependency yields `quartile: 'N/A'` and 0 peer points.

Peer logic

`extractPeerInstitutions()` (lines 401–439) scans the raw appropriations rows directly.

- **Self-exclusion: NO.** There is no filter on the subject institution's unitid, so the institution is a member of its own peer group and is counted in `peerCount` and in the percentile denominator. This differs from the tuition dependency analyzer, which excludes self.

- Filters: sector column must equal `this.sector` (when the column exists); `peerCriteria.carnegieBasic` array; `peerCriteria.hbcuFilter` compared against the literal strings `'1'` and `'2'` (not the `'hbcu-only'` / `'non-hbcu-only'` tokens the other four analyzers use); `peerCriteria.customPeerGroup` unitId whitelist.
- **Statistic used for scoring: a percentile rank** over `peerDependencyValues`, the 2024 dependency of every peer with a usable value, sorted ascending. `percentile = round(count(peers with higher dependency) / n × 100)`.
- `averages.dependency` and `averages.trend` are arithmetic **means**, used only in the narrative comparison text (`calculatePeerComparison`).
- Peer trends are recomputed in the same percentage-point unit as the institution's own (lines 369–379).

2026-08 changes

2026-08-28 (lines 28–64, the redesign block; "Specified by D. Greenstein") — the authoritative statement. Four faults named:

- *"THERE WAS NO PEER FACTOR. All 15 points were absolute. Peer averages were computed and used only for narrative text, and the line printed as 'Overall appropriations risk ranking: Top quartile (excellent compared to peers)' came from calculateOverallPeerRanking, which ignored the peer analysis passed to it and relabelled the institution's own absolute score in quartile language."*
- *"THE TREND WAS RELATIVE TO THE BASE YEAR: ((last - first) / first)* 100 / yearsSpan so 2% -> 4% dependency scored ((4-2)/2)*100/14 = 7.1 and took the maximum trend points, while 60% -> 70% scored 1.19 and was called 'stable'. The institution with far greater absolute exposure growth scored lower. A base year of 0 was also accepted, making the trend Infinity or NaN — both of which fell through every comparison to the final else and took the maximum. The trend is now the change in PERCENTAGE POINTS over the span, which orders those cases correctly."*
- *"thresholds.moderate (60) was never referenced; the branch hard-coded a literal 50, so editing that config value changed nothing."*
- *"dependency.maxPoints was 15 while only 10 was reachable."*

2026-08-28 (line 88, `peerCriteria` default) — *"extractPeerInstitutions dereferences peerCriteria.carnegieBasic, so an omitted argument threw a TypeError that the outer catch turned into a 0/15 'unknown' result — a scoring outcome produced by a missing default rather than by missing data."*

2026-08-28 (line 209, current value) — *"`|| null` was discarding a genuine 0% dependency as missing, which then printed as 'undefined%'."*

2026-08-28 (line 353, peer values retained) — *"the individual peer dependency values are now retained, because the peer factor scores a PERCENTILE RANK rather than a distance from the mean — matching the tuition dependency analyzer."*

2026-08-28 (line 573, `calculateOverallPeerRanking`) — *"this reported the institution's OWN absolute score percentage, relabelled in quartile language, while ignoring the peerAnalysis argument passed to it. Every public report carried a sentence of the form 'Overall appropriations risk ranking: Top quartile (excellent compared to peers)' that had nothing to do with its peers. It now reports the real percentile rank of the institution's dependency within its peer group."*

2026-08-29 (lines 302–316, the two-field split) — *"TWO fields, because they were doing two jobs badly as one. `riskLevel` is the value the framework, the report and layer 2 read. It was a five-value lowercase vocabulary — minimal/limited/moderate/high/critical — on cut-points of its own, so a public report printed 'moderate' in lower case beside every other measure's 'Moderate', and no institution could ever be 'Critical' here in the sense the rest of the tool means it. It is now the house vocabulary on the house ratios: 0.80 Critical, 0.55 High, 0.30 Moderate. The 0.80 matters beyond cosmetics — layer 2's checkCriticalMassEscalation counts any measure at or above 80% of its maximum as critical whatever label it emits, so a label on a different scale made the report disagree with the score it produced. `dependencyBand` keeps the old five-value vocabulary for the narrative text, which describes the DEPENDENCY, not the risk score, and reads correctly."*

2026-08-29 (line 692, error token) — `createErrorResult`'s `riskLevel` *"was lower-case; the house token is 'Unknown'."*

Notes

- The file's own top-of-file header (line 13) still reads *"Framework: 8 points maximum allocation"*, which no longer matches `riskConfig.maxPoints` of 15.
- The 2026-08-29 comment says `dependencyBand` *"describes the DEPENDENCY, not the risk score"*. In the code both `riskLevel` and `dependencyBand` are derived from the same `percentage`, which is `totalPoints / maxPoints × 100` — the risk score. There is no separate dependency-based computation. Reported here as the code stands; the stated intent and the implementation differ.

- `generateRiskInterpretation()` (lines 522–536) bands the raw dependency on its own cut-points (15 / 25 / 40 / 55) that appear nowhere else in the file. Its output is a key finding only.
- `getAllInstitutionsFromData()` (lines 634–654) is dead code and caps at 100 rows.
- Year extraction (`if (yearIndex >= 0 && institutionRow[yearIndex])`, line 180, and the same at line 448) uses a truthiness test. A numeric `0` cell would be dropped; the string `"0"` would not. Which occurs depends on the pipeline's cell types — not determinable from this file. UNCLEAR: whether the upstream loader yields numbers or strings here.

Structural Operating Margin

OPERATING PERFORMANCE & COST STRUCTURE

- **Class**: `UnifiedStructuralMarginAnalyzer`
- **Data key**: `'structural_margin'` (wrapper `UniversalStructuralMarginAnalyzer`, HTML line 1104). A second key is read at scoring time by `getOMDataForComparison`: `window.dataProcessor.getDataForAnalyzer('operating_margin')`, used only by the reserve-dependence divergence kicker. Internal `this.measureName = 'structural_margin'`. Wrapper label string is `'Structural Operating Margin Analysis'`. Dataset regex (`data_processing_pipeline.js` line 33): `/annualized.*structural.*margin/i`. Declared `dataRequirements`: `['annualized structural margin components.xls']`.
- **What it measures**: The header comment states the purpose: "Detects whether an institution's operating 'balance' is funded by recurring operations or by reserve/investment draws. The reported operating margin (and the existing 'adjusted' operating margin) include investment return as operating revenue. An institution can therefore appear to break even or surplus while running a structural operating deficit covered by endowment spending." The report narrative adds: "the structural margin reveals whether recurring operations cover recurring expenses. A persistently negative structural margin indicates reserve-dependent operations." The canonical case named in the header: "Roger Williams University FY2024 … reported margin ~+0.5%, IPEDS-reported margin ~4.3%, but with the endowment spending distribution ($4.87M) stripped out, the structural margin is roughly -2.6% — a Critical-tier signal that the existing analyzers do not surface."
- **Sector differences**:
  - `sectorName`: `'1'` → Public, `'2'` → Private, `'3'` → For-Profit, anything else → Community College.
  - **The header documents two sector-specific formulas**:
  - Privates (sector 2): `structural_margin = (F2D16 - F2D10 - F2E131) / (F2D16 - F2D10)`, where F2D16 = Total revenues and investment return - Total; F2D10 = Investment return - Total (the strip-out); F2E131 = Total expenses.
  - Publics (sector 1) and Community Colleges (sector 4): `structural_margin = (F1B09 + F1B19 - F1B17 - F1C191) / (F1B09 + F1B19 - F1B17)`, where F1B09 = Total operating revenues; F1B19 = Total nonoperating revenues; F1B17 = Investment income (the strip-out); F1C191 = Total expenses and deductions.
  - `this.fieldMap` is set from these in the constructor (line 86): `sector === '2'` → `{ totalRev: 'F2D16', invReturn: 'F2D10', totalExp: 'F2E131' }`; every other sector → `{ opRev: 'F1B09', nonopRev: 'F1B19', invReturn: 'F1B17', totalExp: 'F1C191' }`.
  - **`this.fieldMap` is never read anywhere else in the file** (single occurrence, line 86). The header also says "The analyzer computes the structural margin per year on the fly so the data file can be a clean component pull rather than a pre-computed ratio", but the implementing method `computeMarginSeries` (line 490) carries its own comment — "Read the year-by-year structural margin from **pre-computed columns**. The data file (built by SQL) carries one column per year named either '2010'…'2024' or '*2010'…'\2024' with the structural margin **already calculated upstream**" — and reads bare/asterisk year columns only. The header's component-field expectation and the implementation disagree; the implementation reads pre-computed year columns.
  - **`assessMeasureRisk` contains no sector branch at all.** Every threshold and every point value is identical for public, private, for-profit and community college.
  - The divergence kicker (Factor 4) is **not** scoped to any sector, unlike the Adjusted Operating Margin kicker.
  - Framework points: 8 in all four sectors (`risk_scoring_engine.js` and `framework.json` agree). The HTML wrapper re-asserts `result.analysisResults.maxRiskPoints = 8`.
- **Scoring**:

`maxPoints = this.measureConfig.maxRiskPoints = 8`. `riskPoints` is clamped with `Math.min(riskPoints, maxPoints)`; the four factors sum to at most 4 + 2 + 1 + 1 = **8**, so the clamp never binds. There is **no peer-comparison points factor** in this analyzer.

Factor 1 — 3-year rolling average (max 4 points; declared 50% weight). Ceiling tests, first match wins. Comment: "Rolling average is the primary signal. One-year noise is real."

Condition	Direction	Points	Factor string
`rollingAverage < -3.0`	ceiling	4	Critical 3-year structural margin (…%) — operations consistently below break-even after stripping investment returns
`< 0.0` (and ≥ -3.0)	ceiling	3	High 3-year structural margin risk (…%) — reserve-dependent operations
`< 2.0` (and ≥ 0.0)	ceiling	1	Thin 3-year structural margin (…%) — operations covered but with limited cushion
`>= 2.0`	floor	0	—

Note there is no 2-point band; the ladder steps 4 → 3 → 1 → 0.

Factor 2 — current-year value (max 2 points). Ceiling tests, first match wins.

Condition	Direction	Points	Factor string
`currentValue < -3.0`	ceiling	2	Current year structural margin (…%) indicates active reserve burn
`< 0.0` (and ≥ -3.0)	ceiling	1	Current year structural margin (…%) negative — operations not covering expenses
`< 2.0` (and ≥ 0.0)	ceiling	1	Current year structural margin (…%) thin — limited cushion against reserve dependence
`>= 2.0`	floor	0	—

The -3.0 to 0.0 band and the 0.0 to 2.0 band both award 1 point.

Factor 3 — trend deterioration (max 1 point). Applies only when `measureAnalysis.trends?.direction === 'declining'`.

Condition on `trends.changeRate`	Direction	Points	Factor string
`< -0.5`	ceiling	1	Structural margin declining over analysis period
`>= -0.5`	floor	0	—

This factor is live in this file. `calculateMeasureTrends` here (line 559) explicitly writes both `changeRate: annualizedChange` **and** `annualizedChange: annualizedChange`, so the `changeRate` key that is missing in the Operating Margin and Adjusted Operating Margin trend objects does exist here.

Factor 4 — Reserve-Dependence Divergence Kicker (+1). `calculateReserveDependenceDivergence` compares this measure's own series against the **unadjusted** operating margin series pulled from `getDataForAnalyzer('operating_margin')` (per-row scaled via `rowScale`/`parseScaled` inside `extractOMSeriesFromRow`). It examines exactly the years `'2024'`, `'2023'`, `'2022'`, counts a year as divergent when **both** `om >= 2.0` (floor) **and** `struct <= -2.0` (ceiling), and returns 1 when `divergenceYears >= 2`, else 0. Factor string: "Reserve-dependence divergence: reported operating margin appears healthy but structural margin shows sustained losses (endowment draws are filling the gap)". Header comment: "If existing operating margin reads >= +2% but structural margin reads <= -2% for >= 2 consecutive years, that is the Roger Williams pattern exactly: the reported margin looks fine specifically because reserves are filling the hole. Worth an explicit +1 flag." (Note the header says "consecutive"; the implementation counts any 2 of the 3 named years, not necessarily adjacent.) If `getOMDataForComparison` returns null — no `window.dataProcessor`, no operating-margin dataset, `UnifiedOperatingMarginAnalyzer` undefined, or the institution absent from the OM file — the kicker returns 0 silently.

- **Maximum points**: `this.measureConfig.maxRiskPoints = 8`, declared in the constructor and restated in the header comment ("Max points: 8 (matches existing operating margin / adjusted operating margin); Domain: margin (eligible for critical mass escalation)"). Re-asserted as 8 by the HTML wrapper. Framework: 8 in all four sectors.
- **Risk level mapping**: from **points as a ratio of `maxPoints` (8)**, all floors:

Risk level	Ratio floor	Points floor	Integer points
Critical	≥ 0.80	≥ 6.4	7, 8
High	≥ 0.55	≥ 4.4	5, 6
Moderate	≥ 0.30	≥ 2.4	3, 4
Low	—	< 2.4	0, 1, 2

`riskPercentage = Math.round((riskPoints / maxPoints) * 100)`.

This supersedes the header comment. The header's "RISK FRAMEWORK" block still describes a *value-based* mapping that the code does not implement:

```
  Risk thresholds (3-year rolling average is primary signal):
    >= +2.0%  Low        — operations self-funded
    0..+2.0%  Low-Mod    — thin but positive
    -3..0%    High       — reserve dependent
    < -3.0%   Critical   — burning reserves at clip
```

and `measureConfig.riskThresholds` still carries `{ critical: -3.0, high: 0.0, lowMod: 2.0, low: 999.0 }`. **`measureConfig.riskThresholds` is never read by any method in the file**; the literal values -3.0, 0.0 and 2.0 are hard-coded into Factors 1 and 2 instead, and the risk *level* is derived from points, not from the value. The label "Low-Mod" appears nowhere in executable code.

- **Data-gap behaviour**: two distinct paths, and they differ.
    - `createFallbackResult(unitId, institutionName, errorMessage)` — institution row not found, or `hasValidData` sees fewer than 2 non-null years:

```
    { success: true, measure: 'structural_margin', institution: {...},
      analysisResults: { riskPoints: 0, maxRiskPoints: 8, riskLevel: 'Unknown',
                         isDataUnavailable: true },
      detailedReport: { headline: '<name> structural margin analysis unavailable',
                        keyFindings: ['Data limitation: <errorMessage>'],
                        narrative: '… requires F2D16/F2D10/F2E131 for privates or
                                    F1B09/F1B19/F1B17/F1C191 for publics.' } }
```

`riskLevel` is the string `'Unknown'`; `isDataUnavailable` **is** set (`true`).

    - `createErrorResult(code, message)` — missing `unitId`, or any thrown error in `analyze()`:

```
    { success: false, error: { code, message }, measure: 'structural_margin',
      analysisResults: { riskPoints: 0, maxRiskPoints: 8, riskLevel: 'Unknown' },
      detailedReport: { headline: 'Structural margin analysis error', keyFindings: [message] } }
```

`riskLevel` is `'Unknown'`; **`isDataUnavailable` is NOT set** on this path.

    - The HTML wrapper adds a third: when `structural_margin` data is missing or has fewer than 2 rows, it returns `{ success: false, analysisResults: { riskPoints: 0, riskLevel: 'Unknown' }, detailedReport: { headline: 'Structural margin component data not available' } }` — with no `maxRiskPoints` and no `isDataUnavailable`.

- **Peer logic**: `findAndAnalyzePeerGroup` (line 165) uses the same hand-rolled filter as the margin siblings: peers are drawn from `this.institutions` (rows whose own `sector` cell equals `this.sector`), minus self by `unitId`, then filtered by `peerCriteria.carnegieBasic` (array membership), `peerCriteria.hbcuFilter` and `peerCriteria.customPeerGroup`. No `StandardPeerBenchmarker`. **Peers do not contribute risk points in this analyzer.** The peer statistics are computed (`currentMargin`, `recent3YearAverage`, `rollingAverageTrend`, `volatility`, each with count, mean, median, min, max) and `benchmarkValue` computes a **percentile** and quartile band for display (`peerRanking`, `quartilePosition`, `percentileRank`), but `assessMeasureRisk` never reads `benchmarking`. The report findings quote the peer **mean** ("vs peer average of X%"), unlike the Operating Margin and Adjusted Operating Margin reports, which quote the **median**.
- **Notes**:
    - **The 2026-08-28 shared-margin-vocabulary comment.** The identical block appears in all three margin analyzers, immediately above the risk-level assignment. Quoted in full (from this file, lines 675–690):

> *"2026-08-28 (v49.1): SHARED MARGIN VOCABULARY. The three margin analyzers labelled the same fraction of the same 8-point scale differently: structural margin called 0.75 'Critical' where operating margin and adjusted operating margin called it 'High', so AOM could never emit Critical at all while structural margin emitted 669 of them (43%). The cut-points now match CRITICAL_POINTS_RATIO = 0.80 in checkCriticalMassEscalation, which already treats any measure at or above 80% of its maximum as critical REGARDLESS of its label. That backstop meant AOM was contributing 333 criticals by ratio while reporting 'High' — label and mechanism disagreed in both directions. They now agree. Effect on critical mass: structural margin stops contributing the 218 institutions that sat at 6/8 (75%, below the ratio threshold) and were critical by label only. AOM's contribution is unchanged in substance and now carries a label that matches it."*

`CRITICAL_POINTS_RATIO = 0.80` is confirmed at `risk_adjustment_analyzer.js` line 2881, and used at line 2920 (`if (ratio >= CRITICAL_POINTS_RATIO)`).

- **Per-row scaling deliberately NOT applied here.** The 2026-08-26 decision comment (lines 386–406) is authoritative: "rowScale, parseScaled and `__rejectIsolatedBreaches` are present and are NOT used here. Per-row scaling was applied to this analyzer, measured, and withdrawn, because unlike adjusted operating margin its bound is not real: AOM denominator is TOTAL revenue, so a margin past -100% needs expenses above twice revenue - extreme, and in this file usually a bad cell. STRUCTURAL margin strips investment return from the denominator, so an endowment-dependent college legitimately runs past -100%. Grinnell moved -33.8% -> -113.6% and Lyon -17.5% -> -140.5% under per-row scaling, and those may well be the CORRECT figures - the original rule mixes units inside a row and averages -0.5 read as -50% together with -1.2 read as -1.2%. Both readings are defensible and the data cannot decide between them, so the scoring rule was not changed on a guess. This file has 61 rows stored wholly in percentage points and 156 isolated single-year breaches concentrated at 2010 and 2023-24, where the source query's rolling window shrinks to two years and then one. Fix it in the query; then this analyzer can move to the same rule as AOM." Accordingly `computeMarginSeries` uses `parseNumericValue`, the magnitude rule (`|v| < 1 → ×100`), and `__rejectIsolatedBreaches` is defined but never called.
- **`findColumnIndex` differs from the siblings.** This file tries exact trimmed string equality first and only then falls back to case-insensitive `includes`; the Operating Margin, Adjusted Operating Margin and Net Tuition analyzers use `includes` only.
- The file registers itself on `window` (`window.UnifiedStructuralMarginAnalyzer`); the Operating Margin and Adjusted Operating Margin files do not.

Student Retention

MARKET DEMAND & STUDENT SUCCESS

- **Class**: `UnifiedRetentionAnalyzer` (wrapper: `UniversalRetentionAnalyzer`, wrapper label `'Student Retention Analysis'`, analyzer key `retention`)
- **Data key**: The analyzer does not call `getDataForAnalyzer` itself. The wrapper (line 2008) calls `this.dataProcessor.getDataForAnalyzer('student_retention')`, which maps to `/annualized.*progression.*sfr.*pv.*(and|&).*pu/i`.
- **What it measures**: The percent of students retained from last fall to current fall (`EF{year}D_RET_PCF`, with five alternate column-name spellings tried), read across 2010–2024, scored on both its current level and its total change since the earliest available year.
- **Sector differences**: **None in scoring — retention is deliberately sector-neutral.** The sector-aware `retentionThresholds` block was removed on 2026-08-29 (see below). `this.sector` still filters the peer pool (`extractAllInstitutions` keeps only rows whose SECTOR string equals the caller's) and sets `sectorName` for display. The wrapper overwrites `analysisResults.maxRiskPoints = 6` with the comment `// Both sectors get 6`. `risk_scoring_engine.js` declares 6 points in the public, private, for-profit and community-college blocks alike.
- **Scoring**:

Level component (0–4), scored on the **worse** of two legs (`Math.max(absolutePoints, percentilePoints)`).

Leg A — absolute ladder, gated on `currentRate !== null && currentRate < 65`. Thresholds are **ceilings** (<).

Condition	Points
`currentRate < 50`	4
`currentRate < 55`	3
`currentRate < 60`	2
`currentRate < 65` (and ≥ 60)	1
`currentRate >= 65`	0

Leg B — peer percentile ladder (`retentionLevelFromPercentile`). `percentile` is `round(count of peers the institution beats / peer count * 100)`. Thresholds are **ceilings** (<).

Condition	Points
`percentile < 10`	3
`percentile < 25`	2
`percentile < 50`	1
`percentile >= 50`	0
percentile null / no peer values	0

Trend component (0–2), on `totalChange` in percentage points since the earliest available year. Thresholds are **ceilings** (<).

Condition	Points
`totalChange < -10`	2
`totalChange < -5` (and ≥ −10)	1
`totalChange >= -5`	0

Total is `Math.min(level + trend, 6)`. The components sum to 4 + 2 = 6, which **is** the declared maximum, so the cap is a no-op. Note the level component's own maximum (4) is reachable only through Leg A; Leg B tops out at 3.

- **Maximum points**: `this.riskFramework.maxRiskPoints = 6`. **Reachable** — requires `currentRate < 50` (4 points) **and** `totalChange < -10` (2 points).
- **Risk level mapping**: `determineRiskLevel(totalRiskPoints)`, applied in `analyze()`:

Condition	riskLevel
`riskPoints >= 6`	`High`

Condition	riskLevel
`riskPoints >= 3`	`Moderate`
otherwise	`Low`

`High` is therefore reachable only at exactly 6 of 6. A second, **different** mapping exists inside `assessRetentionRisk` as `retentionRisk` (`>= 4` High, `>= 2` Moderate, else Low) and in `generateEnhancedReport`'s `dataQuality`; `retentionRisk` is **not** published — `analysisResults.riskLevel` uses `determineRiskLevel`.

- **Data-gap behaviour**: `createErrorResult` (missing unitId, or institution absent from the retention dataset) returns `riskPoints: 0`, `maxRiskPoints: 6`, `riskLevel: 'Unknown'`, `isDataUnavailable: true`. If the institution **row exists** but no retention series can be built, `analyze()` does **not** check `retentionAnalysis.dataAvailable` — it proceeds, `assessRetentionRisk` skips the whole scoring block (`if (retentionAnalysis.retentionRate?.dataAvailable)`), and the result publishes `riskPoints: 0`, `riskLevel: 'Low'` with **no** `isDataUnavailable`. **No inner path awards points for missing data.** The wrapper's missing-institution early return publishes `riskPoints: 0, riskLevel: 'High'` — zero points but a `High` label, and no `isDataUnavailable`.
- **Peer logic**: `findAndAnalyzePeerGroup` filters `this.institutions` (already sector-restricted) with an explicit self-exclusion (`inst.unitId !== institution.unitId`), then `peerCriteria.carnegieBasic` (array membership, string-compared), `peerCriteria.hbcuFilter` and `peerCriteria.customPeerGroup`. The **scored** comparison is a **percentile** against `peerAnalysis.statistics.currentRateValues` — every peer's current retention rate, gathered by re-running `analyzeRetentionMetrics` on each peer's row. The **narrative** quotes an arithmetic **mean** (`statistics.currentRate.mean`). `calculateStatistics` computes only `count`, `mean` and the sorted `values` — no median. The institution **is** excluded from its own peer group.
- **2026-08 changes**: one block, 2026-08-29 (v49.1), in the `riskFramework` declaration (lines 52–71) — **removal of a dead sector-aware threshold block**: *"the sector-aware retentionThresholds block that stood here was DEAD CODE - declared, never read. The scorer has always used one ladder for every sector (see assessMeasureRisk: <50 = 4, <55 = 3, <60 = 2, <65 = 1). It is removed rather than wired in, and retention is documented as deliberately sector-neutral."*

The stated reasoning, attributed in the comment to *"Decision, D. Greenstein 29 August 2026"*: *"retention should score the same in both sectors. It is a student-outcome measure, not a financial one - a student who does not return is the same event at a public and at a private. Sector-relative scoring earns its place on measures where the business models genuinely differ, such as appropriations dependency and tuition dependency."*

And on why wiring it in would have broken the measure: *"The dead block would also have broken the measure. It was a 0/2/4/6 ladder from an older design in which level carried 6 points; level now carries 4 and trend 2. And its cut-points (public 85/75/65/60, private 90/80/70/65) sit far above the observed distribution - public median 76%, private 77% - so wiring it in would have made retention fire on roughly 80% of publics and 86% of privates, turning it into another constant-behaving measure of the kind already flagged in the baseline. For the record, the live ladder fires on 11.6% of publics and 18.0% of privates. The sector difference emerges from the data instead of being built into the rule."*

(Undated but present: an inline note in `assessRetentionRisk` recording that *"Previously level and trend both sat behind a gate requiring rate < 65% OR a >5pp decline, so any institution above 65% scored zero regardless of peers. Peer position was computed as a quartile string and never consulted by the scorer."* This is not tagged with a date, so it is not attributed to the August window.)

- **Notes**:
  - `performRetentionBenchmarking` reads `peers.map(p => p.retentionRate?.current)` from `peerAnalysis.validPeers`, but `validPeers` holds the raw institution descriptors `{unitId, name, sector, carnegieBasic, hbcu}` produced by `extractAllInstitutions` — they have no `retentionRate` property. `peerRetentionRates` and `peerRetentionChanges` are therefore **always empty**, `retentionRanking` and `changeRanking` are always `'N/A'`, and the published `analysisResults.peerRanking` is always `'Insufficient data'` (≥ 3 peers) or `'Insufficient peer data'` (< 3 peers). The scorer is unaffected: it reads `peerAnalysis.statistics.currentRateValues` first, which is populated correctly, and falls back to the empty `benchmarking.peerRetentionRates` only if that is absent.
  - The riskFactor string emitted by the percentile leg is hard-coded to read *"…th percentile of public peers"* regardless of the actual sector.
  - `findColumnIndex` matches with `.includes()`, so `EF2010D_RET_PCF` and any header merely containing that substring both match; the year list is tried in declaration order and the first hit wins per year.

- `findInstitutionData` is indexed (a 2026-08-27 performance change, first-wins semantics preserved).
- `standardizePeerFiltering(this)` is called in the constructor when that global exists; what it does is defined outside this file. **UNCLEAR: whether `standardizePeerFiltering` replaces any of the peer methods described above at runtime.**

Student Support

OPERATING PERFORMANCE & COST STRUCTURE

- **Class**: `UnifiedStudentServicesExpenditureAnalyzer` (real analyzer, `unified_student_services_expenditure_analyzer.js`); `UniversalStudentServicesExpenditureAnalyzer` (wrapper, `modular_tool_v48 no data cap.html` lines 1444–1477). Orchestrator key `studentservicesexpenditure`.
- **Data key**: `'student_services_expenditure'`. **This is the only one of the six that calls `getDataForAnalyzer` itself**, in its own constructor at line 21: `this.data = dataProcessor?.getDataForAnalyzer('student_services_expenditure') || []`. The wrapper accordingly passes `this.dataProcessor` rather than a resolved array (line 1459, commented *"PASS THE DATAPROCESSOR, NOT THE DATA"*). Resolved in `data_processing_pipeline.js` line 19 by `/annualized.*pct.*student.*services.*expenditure/i`. Declared `dataRequirements: ['annualized pct student services expenditure.xls']`.
- **What it measures**: student services as a percentage of total expenditures at the latest available year, plus the change since the earliest available year. Header direction: *"Lower student services expenditure = BAD (poor student support) … Too high student services expenditure = BAD (inefficiency)"*. Years collected 2010–2024.

Sector differences

Two ternaries, at lines 63 and 79, both (`this.sector === '1' || this.sector === '4'`). **This is the 2026-08-28 change** — see below. Public and community colleges share one threshold set; sectors `'2'`, `'3'` and anything unrecognised share the other. There is no recommendation-only optimal range in this analyzer (`generateRecommendations` does not exist; `generateEnhancedReport` omits recommendations entirely).

Scoring

Component 1 — current level (lines 361–391). Inefficiency tested first, then the ladder. First match wins.

#	Threshold key	Public / CC	Private and all others	Test	Floor / ceiling	Points
1	`inefficiencyThresholds.wasteful`	25	30	`currentRate >= threshold`	**floor**	**2**
2	`dangerous`	1	2	`currentRate <= threshold`	**ceiling**	**6**
3	`critical`	3	4	`currentRate <= threshold`	**ceiling**	**5**
4	`concerning`	5	7	`currentRate <= threshold`	**ceiling**	**3**
5	`good`	8	10	`currentRate <= threshold`	**ceiling**	**1**
6	—	—	—	otherwise	—	**0**

`excellent` (12 public / 15 private) is defined and **never read**. Resulting bands:

Public / CC current rate	Points	Private current rate	Points
≥ 25	2	≥ 30	2
≤ 1	6	≤ 2	6
> 1 and ≤ 3	5	> 2 and ≤ 4	5
> 3 and ≤ 5	3	> 4 and ≤ 7	3
> 5 and ≤ 8	1	> 7 and ≤ 10	1
> 8 and < 25	0	> 10 and < 30	0

Note the discontinuity at the top: a public institution at 24.9% scores **0**, at 25.0% scores **2**.

Component 2 — trend (lines 393–410). Thresholds are inline literals here, not `riskFramework` values:

#	Test	Floor / ceiling	Points
1	`totalChange < -2`	**ceiling**, strict	**2**
2	`totalChange < 0`	**ceiling**, strict	**1**
3	otherwise (`totalChange >= 0`)	—	**0**

A change of exactly −2 falls to branch 2 and scores 1. A flat series scores 0 and prints *"Positive trend: student services investment increased by 0.0 pts since 2010"*.

`totalPoints = component1 + component2`, then `Math.min(totalPoints, 6)`.

Do the components sum to the declared maximum? No — 6 + 2 = 8 against a declared maximum of 6, so the cap binds. Separately, `riskFramework.factors` (lines 84–87) declares `currentLevel: { weight: 4 }` and `trendDirection: { weight: 2 }`, summing to 6; those weights contradict the 6 and 2 the code actually awards and **`factors` is never read by anything**.

Maximum points

Declared `this.riskFramework.maxRiskPoints = 6` (line 62); the wrapper re-asserts 6 at line 1470 on the `result.success` path. Framework declares 6 in all four sectors. The wrapper does not set `this.maxRiskPoints`, so the base-class catch path would publish 15.

Reachable, and over-reachable — raw total can be 8.

Risk level mapping

`determineRiskLevel(totalRiskPoints)` (lines 581–585), identical to both siblings:

Condition	Level
`totalRiskPoints >= 5`	`High`
`totalRiskPoints >= 3`	`Moderate`
otherwise	`Low`

0–2 Low, 3–4 Moderate, 5–6 High. No `Critical`.

Data-gap behaviour

Path	`success`	`riskPoints`	`riskLevel`	`isDataUnavailable`	In denominator?
`catch` in `analyze()`, including *"Institution data not found for UNITID"*	`false`	0	`'Analysis Failed'`	**`true`**	No
Wrapper early return, `!institution.unitid`	`false`	**3**	`'Moderate'`	*not set*; `maxRiskPoints: 6` **is** set here	No — excluded on `!success`
Row found, fewer than 2 non-null years	`true`	0	`'Low'`	*not set*	**Yes — 0/6**

No path awards points for missing data. The wrapper's `riskPoints: 3` on the missing-institution path never reaches the numerator because the orchestrator requires `result.success`.

Note that `analyzeExpenditureMetrics` here (lines 200–224) differs from its siblings: it only writes `yearlyData[yearStr]` when the column is found, so a year with no column is absent rather than null. That changes `assessDataQuality`'s numerator but not the scoring.

Peer logic

Different plumbing from the two siblings. `performPeerAnalysisWithNewFiltering` (lines 227–260) calls **`this.findAndAnalyzePeerGroup(...)`** — the method installed on the instance by `standardizePeerFiltering(this)` in the constructor (line 92), which delegates to a `StandardPeerBenchmarker` built over `analyzer.institutions`. If `hasEnhancedPeerFiltering` is false it returns `{ success: false, count: 0 }` immediately; **there is no basic-peer fallback in this analyzer**, unlike its siblings. If `peerAnalysis.peerCount === 0` it likewise returns `{ success: false, count: 0 }`.

The subject object passed to the benchmarker is `{ unitId: unitId }` (line 235) — the `unitid` key is not supplied at all, so the second disjunct of the benchmarker's self-exclusion compares against `undefined`.

Is the institution excluded from its own peer group? No. `extractAllInstitutions()` line 162 stores `unitId: this.parseNumericValue(row[unitidIndex])` — a Number — while `unitId` on the subject is `institution.unitid`, a String. `Number !== String` is always true.

Statistics. `analyzePeerMeasureData` (lines 262–297) re-reads each peer from the expenditure file and collects `currentRates` and `totalChanges`, then `calculateStatistics` (lines 558–573) returns `{ count, mean, values }` — **no median**. Nothing in the scoring reads the peer statistics: `assessExpenditureRisk` takes `peerAnalysis` and never uses it. Peer figures appear only in `keyFindings` (mean) and in `performExpenditureBenchmarking`'s rank strings, which use `calculatePeerRanking(..., true)` and are suppressed below 3 valid peers.

`performExpenditureBenchmarking` (lines 299–354) shadows its outer `peerAnalysis` parameter with a loop-local `const peerAnalysis` at line 319, and pre-sorts `peerExpenditureRates` descending before passing them to `calculatePeerRanking`, which does not depend on order.

`analysisResults.currentRate` and `peerCurrentMean` (lines 122–123) are read downstream by `risk_adjustment_analyzer.js` lines 1206 and 1240.

2026-08 changes

One block, `2026-08-28 (v49.1) — COMMUNITY COLLEGE SECTOR GATE` (lines 42–59).

Stated reasoning, quoted: *"WAS WRONG: both threshold ternaries below tested `this.sector === '1'` alone, so sector 4 (community colleges) fell through to the PRIVATE threshold set for expenditureThresholds and inefficiencyThresholds. Community colleges were therefore graded against private thresholds (excellent 15%, good 10%, concerning 7%, critical 4%, dangerous 2%, wasteful 30%) instead of the public ones (12/8/5/3/1, wasteful 25%)."*

Evidence given: *"zero effect on the 2026 baseline, which contains only sectors 1 and 2 — but live and silently wrong for any community-college analysis. Both sibling expenditure analyzers, unified_instructional_expenditure_analyzer.js and unified_institutional_expenditure_analyzer.js, already test (`this.sector === '1' || this.sector === '4'`)."*

Outcome, quoted: *"both ternaries test (`this.sector === '1' || this.sector === '4'`), so community colleges take the public threshold set, matching the siblings. No change to sector 1, 2, 3 or 5 behaviour."* Verified in the code at lines 63 and 79.

Direction of the effect for community colleges: the public set is **more lenient at the bottom** (a CC needs to fall to ≤1% rather than ≤2% to score 6, and to ≤8% rather than ≤10% to score anything at all) and **stricter at the top** (the `wasteful` +2 now triggers at 25% rather than 30%).

Notes

- **This file has no browser export.** It ends at the closing brace of the class (line 593) with no `window.UnifiedStudentServicesExpenditureAnalyzer = …` block, unlike its two siblings and every other analyzer read for this spec. UNCLEAR: whether the tool loads it by a mechanism that does not require the global — the wrapper at line 1459 references the bare identifier `UnifiedStudentServicesExpenditureAnalyzer`, which resolves if the file is loaded as a classic script in the same global scope.
- `riskFramework.factors` declares weights of 4 and 2 that contradict the 6 and 2 the code awards, and is never read.
- Discontinuity at the `wasteful` boundary: 24.9% → 0 points, 25.0% → 2 points (public/CC).
- `excellent` threshold is dead.
- No basic-peer fallback: if the Carnegie or custom-peer filter empties the peer group, the measure reports `success: false` on `peerAnalysis` (which does not propagate to the analyzer's own `success`) and simply prints no peer figures.
- The institution is in its own peer pool; harmless here because peers do not score.
- `assessDataQuality` divides by a hard-coded `totalYears = 14` while collecting 15 years.
- Same substring `findColumnIndex` caveats as the siblings.

Student-Facing Expenditures

OPERATING PERFORMANCE & COST STRUCTURE

- **Class**: `UnifiedStudentFacingExpenditureAnalyzer` (wrapper: `UniversalStudentFacingExpenditureAnalyzer`, wrapper label `'Student-Facing Expenditure per FTE'`, analyzer key `studentfacingexpenditure`)
- **Data key**: The analyzer does not call `getDataForAnalyzer` itself. The wrapper (line 2135) calls `this.dataProcessor.getDataForAnalyzer('student_facing_expenses')`, which maps to `/annualized.*Stu.*facing.*fte.*exp.*pv.*(and|&).*pu/i` (`data_processing_pipeline.js` line 48). The wrapper also passes `this.dataProcessor.institutions` as the analyzer's third constructor argument — which the constructor's signature accepts as `institutionsList` and then **ignores**, building `this.institutions` from the dataset instead.
- **What it measures**: Dollars of student-facing expenditure per FTE, read from bare four-digit year columns 2010–2024. Scored on the latest year's level relative to peers and on the 2010–2024 compound annual growth rate relative to peers. The header frames it as "Higher expenditure = higher fixed costs but may improve retention; Lower expenditure = cost efficiency but may impact student experience."
- **Sector differences**: `measureConfig.riskThresholds` is sector-branched on `selectedSector === '1'`:

Threshold	Public (sector `'1'`)	Everything else (incl. sector `'2'` and `'4'`)
`excellent`	8000	15000
`good`	12000	25000
`moderate`	16000	35000
`poor`	20000	50000

Only `moderate` and `poor` are ever read, and only inside the no-peer-data fallback of Part 1. `dataRequirements` and `measureConfig.dataSource` also branch on sector (`…pu.xls` vs `…pv.xls`). `maxRiskPoints` is 6 for every sector. The peer pool is same-sector only.

Scoring

Part 1 — Current level vs peers (`assessExpenditureRisk`, lines 436–469). Runs when `currentValue !== null` and `peerAnalysis.statistics.currentExpenditures` exists and `median !== null`. `valueComparison = (currentValue - peerMedian) / peerMedian`. Floors, strict inequality:

Condition	Points
`valueComparison > 0.30` (30% above peer median)	3
`valueComparison > 0.15` (and ≤ 0.30)	2
`valueComparison > 0.05` (and ≤ 0.15)	1
`valueComparison <= 0.05` (at or below peer median)	0

Part 1 fallback (lines 459–468), reached when the peer guard fails but `currentValue !== null`. Floors on the absolute dollar level against the sector table:

Condition	Public	Non-public	Points
`currentValue >= thresholds.poor`	≥ $20,000	≥ $50,000	3
`currentValue >= thresholds.moderate` (and < poor)	≥ $16,000	≥ $35,000	2
below `moderate`			0

There is no 1-point band in the fallback.

Part 2 — Growth vs peers (lines 471–501). `institutionTrend` is `calculateTrendChange2010_2024()` = `(last/first)^(1/yearsDiff) - 1`, a decimal CAGR. Runs when that is non-null and the peer growth-rate median is non-null. `trendDifference = institutionTrend - peerMedianTrend`. Floors:

Condition	Points
`trendDifference > 0.05` (5 percentage points of CAGR above peers)	3

Condition	Points
`trendDifference > 0.03` (and ≤ 0.05)	2
`trendDifference > 0.01` (and ≤ 0.03)	1
`trendDifference <= 0.01`	0

Part 2 fallback (lines 502–516), reached when the peer guard fails but `institutionTrend !== null`. Floors on the institution's own CAGR:

Condition	Points
`institutionTrend > 0.06` (6% annual growth)	3
`institutionTrend > 0.04` (and ≤ 0.06)	2
`institutionTrend > 0.02` (and ≤ 0.04)	1
`institutionTrend <= 0.02`	0

Total = Part 1 + Part 2, returned as `Math.min(totalRiskPoints, 6)`.

- **Maximum points**: **6.** `this.measureConfig.maxRiskPoints = 6` (line 65), read into `assessExpenditureRisk` as `maxPoints` (line 434) and used for the clamp and for `detailedReport.maxRiskScore`. Part 1 (max 3) + Part 2 (max 3) = 6, so the cap is reachable. **The success path of `analyze()` (lines 240–247) publishes no `maxRiskPoints` field at all** — `analysisResults` carries only `currentValue`, `growthRate`, `peerRanking`, `riskPoints`, `riskLevel` and `peerComparison`. Only the two failure constructors set `maxRiskPoints: this.measureConfig.maxRiskPoints`. **CONFIRMED for the specific question asked**: the analyzer itself scores out of 6, and the fix lives entirely in the wrapper. The wrapper comment at lines 2151–2160, dated 2026-08-28 (v49.1), states that the 2/3 override is removed, that it "was the sole denominator on every successful run: a 0-6 score reported against a 2- or 3-point maximum", that "the orchestrator clamps points to the cap, so the effect was truncation plus a riskLevel that contradicted the points — 12 publics read 'High' at 2.0/2.0", and that "The framework declares 6 for every sector and the analyzer scores 6; the wrapper now agrees." The wrapper now sets `result.analysisResults.maxRiskPoints = 6` unconditionally (`if (result) { … }`, lines 2161–2164), i.e. on failure results too.
- **Risk level mapping**: lines 519–521, on the **uncapped** running total (the clamp is applied afterwards, at line 524):

Condition	riskLevel
`totalRiskPoints >= 5`	`High`
`totalRiskPoints >= 3` and `< 5`	`Moderate`
`totalRiskPoints < 3`	`Low`

Since the components sum to at most 6, the clamp never changes the outcome here.

- **Data-gap behaviour**:
  - No unitId, or a thrown error → `createErrorResult()` (lines 821–840): `riskPoints: 0`, `maxRiskPoints: 6`, `riskLevel: 'Unknown'`, `isDataUnavailable: true`.
  - Institution not found in the dataset → `createFallbackResult()` (lines 841–859): `riskPoints: 0`, `maxRiskPoints: 6`, `riskLevel: 'Unknown'`, `isDataUnavailable: true`.
  - Institution found but with no usable year columns: `analyzeExpenditureData` still succeeds, `currentExpenditure` is null and `institutionTrend` is null, both parts score 0, and the result publishes `riskLevel: 'Low'` with **no `isDataUnavailable` flag**.
  - Wrapper "institution object missing" path (lines 2126–2132): `riskPoints: 2`, `riskLevel: 'Moderate'`, no `isDataUnavailable` — and then the unconditional `maxRiskPoints = 6` write below it applies to that object too.
- **Peer logic**: `findAndAnalyzePeerGroup()` (lines 136–194) filters the same-sector institution list, removes self, and applies `carnegieBasic` (array membership, string-coerced), `hbcuFilter` and `customPeerGroup`. It then calls `analyzePeerMeasureData()`, which re-reads each peer row and collects the peer's latest expenditure and 2010–2024 CAGR. **Both scoring parts compare against the peer MEDIAN** (`statistics.currentExpenditures.median` and `statistics.expenditureGrowthRates.median`). The peer **mean** is used only in `generatePeerComparison()`

narrative text, and `benchmarkValue()` computes a reversed percentile (lower expenditure = higher percentile) for `peerRanking` / `quartile` display only.

- **Notes**:
  - `measureConfig.growthThresholds` (`excellent 0.02, good 0.05, moderate 0.08, poor 0.12`) and `measureConfig.riskWeights` (`currentLevel 0.6, growthRate 0.4`) are declared and never read. `riskThresholds.excellent` and `riskThresholds.good` are also never read.
  - The console line at 233 prints `${riskAssessment.totalRiskPoints}/5 risk points` — a stale denominator that does not match the framework's 6.
  - `findInstitutionRow()` (lines 780–794), used for **every peer** lookup, resolves the column with `this.headers.findIndex(h => h === 'UNITID')` — an exact, case-sensitive match — and returns `null` if the header is spelled any other way. `findColumnIndex()`, used for the subject institution in `extractInstitutionData()`, is case-insensitive and substring-based. A dataset with a lowercase `unitid` header therefore analyses the institution but silently produces zero peers, pushing both parts to their fallbacks.
  - `assessDataQuality()` calls anything with ≥ 3 year values "High".
  - `calculateGrowthRate()` and `calculateTrend()` are computed into `expenditureAnalysis` but the scorer uses `calculateTrendChange2010_2024()` instead.
  - There is a `console.log` debug block inside the scoring path at lines 474–481 and further logs at 489 and 504.

Student-Faculty Ratio

OPERATING PERFORMANCE & COST STRUCTURE

- **Class**: `UnifiedStudentFacultyRatioAnalyzer` (wrapper: `UniversalSFRAnalyzer`, wrapper label `'Student-Faculty Ratio Analysis'`, analyzer key `sfr`)
- **Data key**: The analyzer does not call `getDataForAnalyzer` itself. The wrapper (line 1784) calls `this.dataProcessor.getDataForAnalyzer('student_retention')` — the **same key as the retention analyzer**, because both measures live in the progression and SFR workbook (`/annualized.*progression.*sfr.*pv.*(and|&).*pu/i`).
- **What it measures**: Students per faculty member, read from `EFyyyyD_STUFACR` columns for 2010–2024. The header describes the risk framing as U-shaped: "Lower ratios = higher fixed costs but potentially better quality; Higher ratios = efficiency focus but potential quality concerns."
- **Sector differences**: **None in scoring**, despite the header comment "SECTOR-ADAPTIVE RISK FRAMEWORK (2 POINTS MAXIMUM)" and the comment "Both sectors get 2 points" on the wrapper. `measureConfig.riskThresholds` and `measureConfig.riskPoints` carry no sector branch. Sector affects only `extractSectorInstitutions()` (peer pool) and `extractInstitutionData()`, which **refuses to analyse an institution whose SECTOR cell does not equal `this.sector`** and returns a fallback result instead.

Scoring

Base — U-shaped level ladder (`assessSFRRisk`, lines 556–571), reading `measureConfig.riskThresholds` and `measureConfig.riskPoints`. Ceilings, evaluated in order:

Condition	Category	Points
`currentValue < 8.0`	Very High Cost Model	2
`currentValue < 12.0` (and ≥ 8)	Premium Cost Model	1
`currentValue < 18.0` (and ≥ 12)	Balanced Efficiency	1
`currentValue < 25.0` (and ≥ 18)	Efficiency Focus	0
`currentValue >= 25.0`	Quality Concerns	2

Peer dispersion adjustment (lines 602–611). **VERIFIED**: an IQR-based dispersion test was added on 2026-08-28 (v49.1). The comment block at lines 574–601 records that the previous test fired on `pct <= 25 || pct >= 75` — a within-peer-group percentile *rank*, which by construction covers roughly half of every peer group — and cites the 2026 baseline: "752 of the 1,417 institutions with SFR data (53%) scored the maximum 2.0/2.0; only 82 scored 0."

The new test:

Condition	Points
`computePeerDispersion(peerVals)` returns non-null **and** `iqr > 0` **and** `Math.abs(currentValue - median) > 1.5 * iqr`	+1
otherwise (including fewer than 4 usable peer ratios, or a zero IQR)	0

`computePeerDispersion()` (lines 439–458) returns `null` when fewer than 4 usable peer ratios are available, and otherwise computes q1/median/q3 with linear-interpolation ("type 7") quantiles. The input array comes from `getPeerCurrentRatios()`, which itself returns `[]` unless `validPeers.length >= 3`. The comment states explicitly that a data gap skips the adjustment and does not fall back to the old rank test.

Trend penalty (lines 613–619). Floor on the relative percent change from first to last year:

Condition	Points
`trends.percentChange > 10`	+0.5
otherwise, or `percentChange` null	0

Combination (lines 622–624): `raw = base + peerAdj + trendAdj`, then `clamped = Math.min(2, raw)`, then `total = Math.round(clamped)`. Because JavaScript's `Math.round` rounds halves up, a base of 0 (Efficiency Focus) plus only the 0.5 trend penalty rounds to **1**.

- **Maximum points**: 2. The analyzer hard-codes `maxRiskPoints: 2` in `analysisResults` (line 180), `maxRiskScore: 2` in `detailedReport` (line 666), and both fallback results. `measureConfig` declares no `maxRiskPoints` field; the 2 comes from the `riskPoints` ladder plus the clamp. The wrapper sets `this.maxRiskPoints = 2` in its constructor and does not overwrite the result.
- **Risk level mapping**: line 626, on the rounded total:

Condition	riskLevel
`total === 0`	`Low`
`total === 1`	`Moderate`
`total === 2`	`High`

- **Data-gap behaviour**:
  - `assessSFRRisk` missing-data branch (lines 542–549), when `currentValue` is null/undefined or `<= 0`: `totalRiskPoints: 0`, `riskLevel: 'Unknown'`, `category: 'No Data'` — and **no `isDataUnavailable` flag**. The success return in `analyze()` copies `riskAssessment.riskLevel` straight through, so this publishes `riskLevel: 'Unknown'`, `riskPoints: 0`, `maxRiskPoints: 2` without the flag.
  - Institution not found, or found in a different sector → `createFallbackResult()` (lines 1099–1137): `riskPoints: 0`, `maxRiskPoints: 2`, `riskLevel: 'Unknown'`, `isDataUnavailable: true`.
  - Thrown error → `createErrorResult()` (lines 1138–1151): `riskPoints: 0`, `maxRiskPoints: 2`, `riskLevel: 'Unknown'`, `isDataUnavailable: true`.
  - Wrapper "institution object missing" path (lines 1776–1782): `riskPoints: 1`, `riskLevel: 'Moderate'`, no `maxRiskPoints`, no `isDataUnavailable`.
- **Peer logic**: `findAndAnalyzePeerGroup()` (lines 358–406) filters the same-sector institution list by `peerCriteria.carnegieBasic` (array `.includes` on the raw value, **not** string-coerced, unlike the other analyzers in this spec), `hbcuFilter` (`'hbcu-only'` keeps `hbcu === '1'`; `'non-hbcu-only'` keeps `hbcu === '2'`), and `customPeerGroup` (`.includes(inst.unitId)`, again not string-coerced), then removes self. It computes no statistics. The **scoring** comparison is against the peer **median and interquartile range** (`computePeerDispersion`). Narrative findings separately build a peer **mean** and a **percentile** on the fly, and require at least 3 peer ratios.
- **Notes**:
  - The base ladder gives 1 point to "Balanced Efficiency" (12–18:1) — which `interpretSFRLevel()` describes as "Optimal balance between operational efficiency and educational quality" — and 0 points to "Efficiency Focus" (18–25:1). The only zero-point band is 18–25:1.
  - `getPeerCurrentRatios()` requires `validPeers.length >= 3` while `computePeerDispersion()` requires at least 4 clean values, so a peer group of exactly 3 can never produce the adjustment.
  - `extractInstitutionData()` (lines 205–226) resolves columns with `this.headers.indexOf('UNITID')` and `this.headers.indexOf('SECTOR')` — exact, case-sensitive matches — while `findColumnIndex()` used everywhere else in the file is case-insensitive. It also scans `this.data` including row 0 (the header row).
  - `performSFRBenchmarking()` (lines 510–535) returns literal placeholder strings on the success path: `quartile: 'Calculated in findings'`, `peerRanking: 'Calculated in findings'`, `comparison: 'Peer comparison available in detailed findings'`. `analysisResults.peerRanking` is therefore the string `'Calculated in findings'`.
  - `analyzePeerMeasureData()`, `calculateRobustStatistics()` and `getQuartileFromPercentile()` are defined and never called on the analysis path.
  - The console line at 167 prints `${riskAssessment.totalRiskPoints}/2`, consistent with the maximum.
  - No entry in the orchestrator's `RA_CANONICAL` map; `result.measure` would be `'student_faculty_ratio'`.

Transfer Out Rate

getAnalyzerDisplayName has no entry — publishes as Transferoutrate

- **Class**: UnifiedTransferOutRateAnalyzer (wrapper: UniversalTransferOutRateAnalyzer, wrapper label 'Transfer Out Rate Analysis', analyzer key transferoutrate)
- **Data key**: The analyzer does not call getDataForAnalyzer itself. The wrapper (line 2618) calls this.dataProcessor.getDataForAnalyzer('transfer_out_rate'), which maps to /annualized.*transfer.*out.*rate/i (data_processing_pipeline.js line 45).
- **What it measures**: The percentage of students who transfer to other institutions, read from bare four-digit year columns 2010–2024. Higher is worse. The latest year's rate is scored, plus the rise in percentage points from the earliest to the latest year.
- **Sector differences**: **None.** No threshold, point value or maximum varies by sector. Sector affects only the peer pool and narrative text. Not on the orchestrator's CC-excluded list, and not on the HTML CC_EXCLUDED_KEYS display filter, so it runs and displays for sector '4'.

Scoring

Level ladder (LEVEL_BANDS, lines 587–592). Floors — points are awarded when the rate is at or above the threshold. Evaluated in order, first match wins:

Condition	Points
currentValue >= 40.0	6.0
currentValue >= 30.0 (and < 40)	4.5
currentValue >= 20.0 (and < 30)	3.0
currentValue >= 12.0 (and < 20)	1.5
currentValue < 12.0	0

VERIFIED: this ladder is **unconditional**. assessMeasureRisk() reaches it with no trend precondition; the only guard above it is the data-gap check on currentValue (lines 571–584). The 2026-08-28 (v49.1) comment block at lines 519–562 records the removal of the previous isHighRate && isIncreasing conjunction gate, and cites the 2026 baseline evidence: 237 institutions above 30%, of which 105 scored 0/8 and published as "Low" (47 decreasing, 44 stable, 14 with no computable trend), the highest zero-scoring rate being 76%.

Trend modifier (TREND_BANDS, lines 595–598), on overallChange in percentage points (NOT percentChange; the comment at lines 559–562 explains the choice). Floors, strict inequality, first match wins, applied only when overallChange is finite and trends.direction !== 'insufficient-data':

Condition	Points
overallChange > 5.0	2.0
overallChange > 2.0 (and ≤ 5.0)	1.0
overallChange <= 2.0, or trend unusable	0

Total = currentRatePoints + trendPoints. There is no explicit Math.min clamp; the two ladders sum to at most 8 by construction.

Fraction of the scale reachable (as requested): the level ladder alone reaches **6 of 8 = 75%** of the declared maximum. With the trend modifier the full **8/8 is reachable** (rate ≥ 40% and a rise of more than 5 percentage points), which is what the comment at lines 552–554 asserts. The attainable totals are the 15 discrete values {0, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 8}; **7.5 is not attainable** (level 6 + trend 1 = 7, level 4.5 + trend 2 = 6.5, and there is no 1.5-point trend band).

- **Maximum points**: this.measureConfig.maxRiskPoints = 8 (line 58), published as analysisResults.maxRiskPoints; the wrapper re-asserts 8 on the success path (line 2645).
- **Risk level mapping**: lines 625–627, on the total:

Condition	riskLevel
riskPoints >= 6	High

Condition	riskLevel
`riskPoints >= 3 and < 6`	`Moderate`
`riskPoints < 3`	`Low`

- **Data-gap behaviour**:
  - `assessMeasureRisk` data-gap branch (lines 571–584), for a null/undefined/NaN current rate: `totalRiskPoints: 0`, `maxRiskPoints: 8`, `riskLevel: 'Unknown'`, `isDataUnavailable: true`, `riskPercentage: null`.
  - No unitId → `createErrorResult('MISSING_UNITID', …)`: 0 points, `'Unknown'`, `isDataUnavailable: true`.
  - Institution not found, or zero usable year values → `createFallbackResult()`: 0 points, `'Unknown'`, `isDataUnavailable: true`.
  - Wrapper paths (institution missing, data file not loaded, thrown error): 0 points, `maxRiskPoints: 8`, `'Unknown'`, `isDataUnavailable: true`.
  - Note that the success return in `analyze()` (lines 215–228) copies only `riskAssessment.riskLevel`, not `isDataUnavailable`, into `analysisResults`. In practice the data-gap branch of `assessMeasureRisk` is unreachable from the success path, because `analyzeMeasureData` returns `dataAvailable:false` (→ fallback result) whenever `measureData` is empty.
- **Peer logic**: Peers are formed exactly as in the 150% grad rate analyzer (same-sector rows, minus self, `carnegieBasic` array membership, `hbcuFilter`, `customPeerGroup`). `benchmarkValue()` computes a reversed **percentile** (`count(peers with HIGHER transfer out rate) / peer count`, so higher percentile = better) and `calculateStatistics()` yields mean/median/min/max. **None of this feeds the score.** `assessMeasureRisk()` accepts `benchmarking` and `peerAnalysis` as parameters and never references either. Peer output appears only in `peerRanking`, `quartilePosition`, `percentileRank`, `peerComparison` and the narrative.
- **Notes**:
  - `getAnalyzerDisplayName()` has **no key** for `'transferoutrate'`. The fallback branch title-cases the first letter, so the export label is the literal `Transferoutrate`.
  - `measureConfig.riskThresholds` (`excellent:10, adequate:20, moderate:30, critical:40`) is not used by the scorer; those values are duplicated as literals in `assessRetentionStatus()`, which only produces the `retentionStatus` string. Note the scorer's level ladder uses 40/30/20/**12**, and 12 has no counterpart in `riskThresholds`.
  - `measureConfig.riskWeights` (`currentRate 0.5, trend 0.3, peerComparison 0.2`) is declared and never read.
  - `extractInstitutionData()` here is a plain linear scan; the `__unitRow` index added to the 150% grad rate analyzer on 2026-08-26 was not applied to this file.
  - `extractMeasureData()` drops any parsed value `< 0`.
  - No entry in the orchestrator's `RA_CANONICAL` map.

Tuition Dependency

REVENUE DEPENDENCE & PRICING

- **Class**: `UnifiedTuitionDependencyAnalyzer` (real analyzer); `UniversalTuitionDependencyAnalyzer` (wrapper, lines 2179–2217). Orchestrator key `tuitiondependency`.
- **Data key**: `'tuition_dependency'`. Called in the wrapper at line 2196. Resolved in `data_processing_pipeline.js` line 30 by `/annualized.*tuition.*dependency.*pv.*(and|&).*pu/i`. The analyzer's own `measureConfig.dataSource` string is `'annualized tuition dependency TUFEPC'`.
- **What it measures**: exposure to a single revenue source — tuition and fees as a share of core revenues. The header states: *"High dependency on tuition is the risk. Its mirror is SimpleAppropriationsAnalyzer, which does the same job for state appropriations at publics; the two are now built to one shape — same three factors, same band form, same peer method, same vocabulary."*

Sector differences

`maxPoints` (line 27) is `(selectedSector === '1' || selectedSector === '4') ? 8 : 18`. All three band sets branch on the same test, so public and community college share one ladder and everything else (private, for-profit) shares the other.

The wrapper (line 2211) overrides the published `maxRiskPoints` to `this.sector === '1' ? 8 : 18` — which does **not** match the analyzer for community college. See Defects.

Scoring

The band helper (lines 553–557) is `for (const b of bands) { if (v >= b[0]) return b[1]; }` — **every level and trend threshold is a floor: the band fires when `value >= X`.**

`valueScale: 1` (line 57) with `toPct = v => v * (100 / cfg.valueScale)`. The underlying series is a **ratio (0–1)**; both the level and the trend are multiplied by 100 before banding, so all thresholds below are in percent / percentage points.

Factor 1 — absolute dependency level (`levelBands`, lines 60–62):

Condition	public / CC	private (and all other sectors)
dependency >= 85%	5	11
dependency >= 75%	5	9
dependency >= 70%	4	9
dependency >= 65%	4	6
dependency >= 55%	2	6
dependency >= 45%	1	3
dependency >= 40%	0	3
below the lowest floor	0	0

Stated as the raw ladders: public/CC `[[75,5],[65,4],[55,2],[45,1]]`; private `[[85,11],[70,9],[55,6],[40,3]]`.

Reference distributions recorded in the config (lines 63–64): C20 (Small Masters) P50 = 0.60, P75 = 0.70, P90 = 0.80; C21 (Liberal Arts) P50 = 0.35, P75 = 0.50, P90 = 0.60.

Factor 2 — trend, change in dependency in PERCENTAGE POINTS over the span (`trendBands`, lines 67–69). Span is first to last non-null year in 2010–2024.

Condition	public / CC	private
rise >= +10 pp	1	3
rise >= +5 pp	0	2
otherwise	0	0

Factor 3 — peer position, percentile rank within the peer group (`peerBands`, lines 72–74, applied at lines 589–601). Driven by the `quartile` string from `benchmarkValue()`, not by the raw percentile.

Quartile (percentile band)	public / CC	private
`Bottom quartile` (percentile < 25 — worst dependency)	2	4
`Below median` (25 <= percentile < 50)	1	2
`Above median` (50 <= percentile < 75)	0	0
`Top quartile` (percentile >= 75 — best)	0	0
`N/A`	0	0

Note the percentile is defined so **higher is better**: `benchmarkValue()` (lines 411–446) counts `peerValues.filter(v => institutionValue < v)` — peers with *higher* dependency — so a low percentile means high dependency.

Do the components sum to the declared maximum? Yes. Public/CC: 5 + 1 + 2 = **8**. Private: 11 + 3 + 4 = **18**. The header records the previous failure: *"It also summed to 19 against a private cap of 18, so the cap clipped."*

Header-stated weighting: *"Level is prominent, peer is meaningful, trend is light: public / CC (8): level 5 trend 1 peer 2; private (18): level 11 trend 3 peer 4."*

Maximum points

`this.measureConfig.maxRiskPoints` — **8 for sectors `'1'` and `'4'`, 18 for everything else.** Published as `riskAssessment.maxRiskPoints`.

Reachable? Yes for both. The maximum requires dependency at or above the top floor, a rise of at least +10 pp, and a bottom-quartile percentile — mutually compatible.

Risk level mapping

Lines 606–610, computed from ratios of `maxPoints`:

Condition	riskLevel
riskPoints >= maxPoints × 0.75	`High`
riskPoints >= maxPoints × 0.50	`Moderate`
riskPoints >= maxPoints × 0.25	`Low-Moderate`
otherwise	`Low`

In points: public/CC (max 8) — Low 0–1, Low-Moderate 2–3, Moderate 4–5, High 6–8. Private (max 18) — Low 0–4, Low-Moderate 5–8 (cut at 4.5), Moderate 9–13 (cut at 9), High 14–18 (cut at 13.5).

This measure never emits `Critical`. `Low-Moderate` is a token unique to this analyzer among the five.

Data-gap behaviour

- **Institution row not found, or data < 20% complete** (`extractInstitutionData`, lines 178–183) → `createFallbackResult()` (lines 899–940): `success: true`, `riskPoints: 0`, `riskLevel: 'Unknown'`, **`isDataUnavailable: true`**, `maxRiskPoints` the sector max, `currentValue: 0`. Excluded from the denominator.
- **Fewer than 2 valid year values** → `analyzeMeasureData` returns `dataAvailable: false` → the same `createFallbackResult()` path.
- **Missing `unitId`, or a thrown error** → `createErrorResult()` (lines 877–897): `success: false`, `riskPoints: 0`, `riskLevel: 'Unknown'`, **`isDataUnavailable: true`**, `maxRiskPoints: this.measureConfig?.maxRiskPoints || (this.sector === '1' ? 8 : 18)`.
- **Wrapper-level gap** (line 2186–2192): institution object missing `unitid` → `{ riskPoints: 6, riskLevel: 'Moderate' }`, **no `isDataUnavailable`, no `maxRiskPoints`**. This path awards 6 points for missing data and stays in the denominator.
- **No path inside the analyzer awards points for missing data.** Individual factors are simply skipped when their input is null.

Peer logic

`findAndAnalyzePeerGroup()` (lines 464–514) over `this.institutions`, which `extractAllInstitutions()` (lines 725–768) has restricted to rows whose `sector` column equals `this.sector`.

- **Self-exclusion: yes** (line 468, on both `unitId` and `unitid`). The subject is passed in as `{ unitId }` by `performPeerAnalysisWithNewFiltering`.
- Filters: `carnegieBasic` array; `hbcuFilter` — `'hbcu-only'` requires `hbcu === '1'`, `'non-hbcu-only'` requires `hbcu === '2'` (the default when no HBCU column exists is `'2'`); `customPeerGroup` unitId whitelist, compared **without** `.toString()` on the member.
- **Statistic used for scoring: a percentile rank**, not a mean or median. `analyzePeerMeasureData()` collects each peer's current dependency; `calculateStatistics()` computes count / mean / median / sorted values; `benchmarkValue()` then computes `percentile = round(count(peers with higher dependency) / n × 100)` and maps it to a quartile string. Only the quartile string feeds the score.
- `calculateStatistics()` also produces a **mean** (`currentDependency.mean`, `overallChange.mean`) used in the narrative findings, and a **median**, used nowhere in scoring.
- `dataQuality` is `'Good'` at >= 5 peers, else `'Limited'`. `assessConfidence()` returns `"Medium - Limited Peer Data"` below 5 peers.

2026-08 changes

2026-08-28 (lines 35–56, the config block) — the threshold-direction fix. *"Thresholds are FLOORS, expressed in PERCENT. The previous config named each threshold for the band BELOW it — `low: 0.55, // 45-55% = low risk` — i.e. ceilings, while the code tested `currentValue >= thresholds.low` and awarded the low points. Read as floors, the whole ladder sat one slot too high: the 'low' award fired at 55% rather than 45%, the 45-55% band scored nothing at all, `minimal` was never referenced, and `high` had to be set equal to `veryHigh` to close the top — which made the `high` branch (`>= 0.75 && < 0.75`) unreachable and skipped its 5-point (public) / 13-point (private) award entirely."*

The block is explicit that the numbers themselves did not move: *"The cut-points themselves — 45/55/65/75 public, 40/55/70/85 private — are unchanged. They are simply attached to the right bands now, and read in percent so this analyzer and the appropriations analyzer, which are mirror images of one another, use one legible scale."* `valueScale` was added to record the units of the underlying series *"so the shared ladder shape can be identical across both analyzers: this measure stores dependency as a ratio (0-1), appropriations stores percent."*

2026-08-28 (lines 520–544, the redesign block; "Specified by D. Greenstein") — the weighting rebuild. *"Integer ladders summing exactly to the sector maximum. The previous version derived every award as a fraction of a fraction of maxPoints, which on small maxima collapsed tiers into each other — Math.round(1* 0.5) === 1 meant a public institution scored the same peer points for 'Below median' as for 'Bottom quartile', and the same trend points for a 6pp rise as for a 12pp rise. It also summed to 19 against a private cap of 18, so the cap clipped." *And on the peer weighting:* "Peer comparison was the LIGHTEST factor at 1 of 8 and 2 of 18, not the heaviest as the design intended."*

Notes

- `measureConfig.measureName` and `.description` are built as `selectedSector === '1' ? 'Public' : 'Private'`, so community college (`'4'`) is labelled *"Private Universities"* in those strings. They surface only in `extractInstitutionData`'s error text.
- The `trendDirection` and `trends.severity` fields use raw ratio thresholds (`Math.abs(overallChange) >= 0.05` / `>= 0.10`, i.e. 5 pp / 10 pp) and are reported, not scored.
- `assessDependencyRisk()` (lines 623–626) is a compatibility shim that forwards to the peer-aware version with `{ success: false }`; nothing in the analyzer calls it.
- `performMeasureBenchmarking()` (lines 869–874) is dead code.
- The recent-trend series (`recentData`, years >= 2020, `recentTrend`) is computed and reported but **not scored**.

Unrestricted Net Assets per Student FTE

LIQUIDITY & BALANCE SHEET

- **Class**: `UnifiedUnrestrictedNetAssetsAnalyzer` (real analyzer); `UniversalUnrestrictedNetAssetsAnalyzer` (wrapper, lines 1701–1775). Orchestrator key `unrestrictednetassets`.
- **Data key**: `'unrestricted_net_assets'` (wrapper line 1709). Resolved in `data_processing_pipeline.js` line 42 by `/annualized.*unrestricted.*net.*asset/i`. The wrapper additionally fetches `'enrollment'` (line 1728) and passes it to the constructor as `{ enrollment: enrollmentData || [] }` — the per-FTE denominator.
- **What it measures**: *"The spendable cushion — unrestricted resources per student. Both absolute and peer-relative by design."* Scored as the percentile of unrestricted net assets per FTE within the peer group, plus the percentile of the change in that figure over a sector-appropriate window.

Sector differences

This is the sharpest sector split of the five, and the header gives the reasoning at length:

> *"GASB 68 (FY2015) and GASB 75 (FY2018) moved net pension and OPEB liabilities onto the balance sheet. The share of publics with negative unrestricted net assets went 12% -> 50% in FY2015 and 50% -> 66% in FY2018, with no operational change; the private series is flat across all fifteen years… The level, the sign, and any negative streak are therefore, for a public, largely a statement about its STATE's pension funding rather than its own management — two identically-run publics in differently-funded systems will differ sharply. What survives the accounting shift is POSITION RELATIVE TO PEERS, because every public took the same hit in the same year. So: publics and community colleges are scored peer-relative only, over a post-GASB window; privates keep the absolute tests, where negative unrestricted net assets are 3% of the sector and mean what they appear to."*

The supporting evidence table transcribed from `extractNetAssetsData` (lines 320–327), median unrestricted net assets and share negative:

Year	Public median	Public % negative	Private median	Private % negative
2014	+$10,418,686	12.1%	+$30,449,458	2.5%
2015	−$3,001	50.0%	+$30,054,197	3.4%
2017	−$819,514	52.8%	+$33,373,982	3.6%
2018	−$11,642,274	65.6%	+$35,081,155	3.3%
2024	+$565,791	48.4%	+$38,963,528	3.8%

2015 is annotated *"GASB 68, pensions"* and 2018 *"GASB 75, OPEB"*.

`isPublicLike = (this.sector === '1' || this.sector === '4')` selects both the point allocation and whether factors 3 and 4 apply. `trendStartYear()` (lines 648–652) returns **2019 for public/CC** and **2010 for everything else** — *"Post-GASB window for publics and community colleges; matches UnifiedPrimaryReserveAnalyzer.TREND_START_YEAR. Privates use the full span."*

Scoring

All comparisons are percentiles. `percentileOf()` (lines 661–667): `round(((below + 0.5 × equal) / n) × 100)`, where `below` counts peers the institution exceeds. **Higher UNA per FTE is better: 0 = worst, 100 = best.** All percentile thresholds below are **ceilings on the percentile (`percentile < X`)**, i.e. floors on risk. The tie-splitting is deliberate:

> *"Ties are split, not counted against the institution: an institution sitting exactly on a cluster of identical peer values lands mid-cluster rather than at percentile 0. Without this, a peer group in which many institutions share a value — common where a series is flat or rounded — puts every one of them in the bottom decile and hands them all maximum risk points."*

Factor 1 — level: UNA per FTE percentile (lines 674–710). `levelMax` is 3 for public/CC, 2 for private.

Condition	public / CC (levelMax 3)	private (levelMax 2)
percentile < 10	3	2
percentile < 25	2 (= 3 × 2/3)	1.333… (= 2 × 2/3)
percentile < 50	1 (= 3 × 1/3)	0.667… (= 2 × 1/3)
percentile >= 50	0	0

Factor 2 — trend: change in UNA per FTE over the window, percentile (lines 712–739). `trendMax` is 2 for public/CC, 1 for private. The change is **absolute dollars**, not a percent — *"the percent form required a positive base year and was undefined for most publics."*

Condition	public / CC (trendMax 2)	private (trendMax 1)
percentile < 10	2	1
percentile < 25	1.333… (= 2 × 2/3)	0.667… (= 1 × 2/3)
percentile < 50	0.667… (= 2 × 1/3)	0.333… (= 1 × 1/3)
percentile >= 50	0	0

When `trendPct` is null the factor is skipped with a note; no points are awarded.

Factors 3 and 4 — absolute tests, PRIVATE SECTOR ONLY (lines 741–755). *"Excluded for publics: a negative streak there dates from FY2015/FY2018 and reflects pension and OPEB recognition, not chronic operating weakness."*

Condition	private	public / CC
`maxNegativeStreak >= 3` (3+ consecutive years of negative UNA)	1	0 — reported as a finding only
`bailoutDetected`	1	0 — reported as a finding only

`bailoutDetected` is set (lines 367–373) when any year-over-year pair has `prev < 0 && (curr - prev) / Math.abs(prev) > 0.5` — a jump of more than 50% of the prior magnitude out of a negative position.

Do the components sum to the declared maximum? Yes for both. Public/CC: 3 + 2 = **5**. Private: 2 + 1 + 1 + 1 = **5**.

Header-stated allocation: *"public / CC (5): level percentile 3, trend percentile 2; private (5): level percentile 2, trend percentile 1, sustained negative streak 1, bailout pattern 1."*

Component scores are not integers. The `× 2/3` and `× 1/3` multipliers produce 1.333…, 0.667… and 0.333… in both sectors. These are published verbatim as `riskPoints`; nothing rounds them.

Maximum points

`this.riskConfig.maxPoints = 5` for **all** sectors (line 35). The analyzer's `analysisResults` object does **not include a `maxRiskPoints` field at all**; the wrapper supplies it (`maxRiskPoints: 5`, plus a redundant `maxPoints: 5`, lines 1740–1741).

Reachable? Public/CC: yes — level percentile < 10 and trend percentile < 10 give 3 + 2 = 5. Private: yes in principle — 2 + 1 + 1 + 1 = 5 requires bottom-decile level, bottom-decile trend, a 3-year negative streak and a bailout jump simultaneously. All four are compatible (a bailout requires a prior negative year, which is consistent with a streak), so 5/5 is reachable but rare.

Risk level mapping

Lines 759–761, on raw points (not a percentage):

totalPoints	riskLevel
exactly 0	`Low`
> 0 and <= 2	`Moderate`
> 2 and <= 4	`High`
> 4	`Critical`

Because points are fractional, the practical boundaries fall between tiers: e.g. a public institution at level 1 + trend 1.333… = 2.333… reads `High`.

`unaStatusFromPercentile()` (lines 598–606, `critical` < 10, `weak` < 25, `fair` < 50, else `strong`) is defined but **never called**.

Data-gap behaviour

- **Level percentile cannot be formed** — no FTE for the institution, no `current2024`, or no peer with both a value and an FTE (lines 681–693). Returns `totalPoints: 0`, `riskLevel: 'Unknown'`, **`dataUnavailable: true`**, `levelPercentile: null`, `trendPercentile: null`. Published at line 183 as `isDataUnavailable: true`. **No points**

awarded. This is the header's stated fix: *"Missing peer or enrolment data previously added 1 point on the level factor and 0.5 on the trend factor — a data gap scored as risk. It now returns dataUnavailable so the orchestrator drops the measure from both the numerator and the denominator."*

- **Trend percentile cannot be formed**: the factor is skipped with a note; the level factor still scores.
- **Institution not found in the net assets file, no year values at all, or a thrown error** → `createErrorResult()` (lines 964–978): `success: false`, `riskPoints: 0`, `riskLevel: 'Unknown'`, `currentValue: null`. **`isDataUnavailable` is NOT set, and no `maxRiskPoints` is set.** See Defects.
- **Wrapper-level gaps** (lines 1709–1720 and 1766–1774). Both award points and neither sets `isDataUnavailable`:
  - `unrestricted_net_assets` dataset absent → `{ success: false, riskPoints: 3, riskLevel: 'Moderate' }`
  - any thrown error → `{ success: false, riskPoints: 3, riskLevel: 'Moderate' }`

Because the wrapper overrides `analyze()` rather than `performSectorAnalysis()`, these `success: false` values do reach the orchestrator, which excludes them from both numerator and denominator. The published `riskPoints: 3` still appears in the result object.

Peer logic

`findAndAnalyzePeerGroup()` (lines 544–594) over `this.institutions`, restricted by `extractAllInstitutions()` to rows whose `sector` column equals `this.sector`.

- **Self-exclusion: yes**, explicitly (lines 555–557).
- Filters: `carnegieBasic` array; `hbcuFilter` (`'hbcu-only'` keeps `hbcu === '1'`, `'non-hbcu-only'` drops `hbcu === '1'`; default when the column is absent is `'2'`); `customPeerGroup` unitId whitelist.
- **Statistics** (`performPeerBenchmarking`, lines 402–465). `averages.current2024` and `averages.percentChange` are **medians**, computed by `medianOf()` — *"median — GASB 68/75 tails make the mean unusable"*. The field name says `averages`; the values are medians.
- **Scoring uses neither.** Factors 1 and 2 build their own per-FTE arrays from `peerValues.current2024Pairs` and `peerValues.windowedChangePairs` and take a percentile. The medians appear only in the narrative findings.
- `current2024Pairs` carries `{unitId, value}` deliberately (lines 448–453): *"Do NOT collapse it back to a bare array indexed against validPeers: validPeers holds every peer, while the value arrays only receive a push when a peer has data. Any peer missing data shifts every later index, silently pairing values with the wrong institution's FTE. This corrupted ~1/3 of the C22 cohort before it was caught."*
- **FTE denominator** (`buildEnrollmentMap`, lines 474–528). Preferred: IPEDS 12-month FTE (`EFIA<year>_EFTEUG` + `EFIA<year>_EFTEGD`, latest year present). Fallback: the headcount column matching `/^((?:19|20)\d{2})\s+total$/`, latest year. Rows with a non-finite or non-positive value are skipped. Built once in the constructor — *"the peer loop runs ~60x per institution across ~1,100 institutions per run."*

2026-08 changes

2026-08-27 (line 225, indexing) — *"This .find() ran once per peer in the peer-group loop, scanning the whole dataset each time — quadratic in cohort size for a ranking run. The index reproduces the same predicate, including the header-row skip, and keeps first-wins semantics to match .find()."*

2026-08-28 (lines 136–146, `una_per_fte_normalized` removed) — *"una_per_fte_normalized (= unaPerFte / peerUnaPerFteMedian) IS GONE. It inverted whenever the peer median was negative — public Carnegie 18, 20, 21 and 22 — so the institutions with the thinnest cushions produced the highest index values, and the two layer-2 gates reading it (checkFinancialDistress and calculateP5_UNARisk) had the signal backwards for roughly 267 publics. A ratio cannot be made safe against a sign change in its denominator. The percentile below has no denominator; 0 is the weakest and 100 the strongest, in every peer group, whatever the sign of its members."* And: *"The old field name is deliberately NOT retained as an alias: a stale consumer should fail loudly on a missing field rather than quietly read a broken one."* The replacement field is `una_per_fte_percentile`.

2026-08-28 (lines 312–334, windowed absolute change) — *"percentChange above requires first2010 > 0 and divides by it. For publics that fails outright — the 2010 value is often negative — and where it does compute it spans both GASB adoptions, so it measures an accounting change rather than the institution… So the trend is now an ABSOLUTE change in dollars over a sector-appropriate window — no division, no sign dependence — and is scored by percentile against peers, who took the same accounting hit at the same time."*

2026-08-28 (line 420, paired peer values) — the `{unitId, value}` pairing described above, *"so it can be converted to a per-FTE figure without the index-drift hazard."*

2026-08-28 (lines 608–647, the redesign block; "Specified by D. Greenstein") — the sector split, the percentile-everywhere rule and the data-gap fix, quoted above.

2026-08-28 (lines 882–893, scored-or-not notes on findings) — *"these two findings must now say whether they were SCORED. The scoring became sector-aware — publics and community colleges are no longer charged for a negative streak or a bailout pattern… Without this note a public report showed 'UNA negative for 8 consecutive years' and 'Detected bailout pattern' beside a score of 0/5, and the words contradicted the number."*

2026-08-29 (lines 846–854, sign-crossing narrative) — *"a percent change is only reportable when the series does not cross zero. CSU San Bernardino went from a positive 2010 balance to -$721.0M in 2024, and the arithmetic dutifully produced 'decreased by 1951.9%' — a figure no reader can interpret, printed beside a peer median of -18.5% as though the two were comparable. The scoring already stopped using this ratio…; the finding text had not caught up. When the sign flips, state the dollars. They are unambiguous and the peer percentage is simply not commensurable, so it is not printed alongside."*

Notes

- `current2024` is `values[values.length - 1]` — the **last available year**, not necessarily 2024. The field name, the published key `percentChange2010to2024`, and the finding text *"in 2024"* all assert 2024 regardless. `firstYear` is published and is used correctly in the narrative; there is no equivalent `lastYear`.
- `percentChange` requires `first2010 > 0` and is `null` otherwise; it is reported, never scored.
- `formatCurrency()` (lines 956–962) tests `value >= 1000000` etc. without absolute value, so negative amounts fall through to `` `$${value.toLocaleString()}` `` and print in full (e.g. `$-721,000,000`) rather than as `-$721.0M`.
- `calculatePeerRanking()` (lines 774–831) compares raw dollar totals against the peer **median total**, with four sign cases. It is the string published as `peerRanking`, and it is computed on a different basis from the score (raw dollars vs per-FTE percentile), so the two can disagree.
- Lines 115–118 test `typeof riskAssessment !== 'undefined'` against an identifier that is never declared anywhere in the file; the expression always resolves to `totalRisk`.
- `window.__UNA_LAST__`, `window.una()`, `window.__UNA_HEADERS__`, `window.__UNA_ROW__`, `window.__UNA_KEYS__` and the IIFE at lines 980–1030 are development console helpers.

Viability Ratio

LIQUIDITY & BALANCE SHEET

- **Class**: `UnifiedViabilityAnalyzer`
- **Data key**: `'viability'` (wrapper `UniversalViabilityAnalyzer`, HTML line 2070). Two further keys are pulled as `adjustmentInputs` for the restriction-adjusted ratio: `'unrestricted_net_assets'`, and `'debt_revenue'` with fallback `'annualized_debt_and_d_r_consist_0_pv___pu'`. Internal `this.measureName = 'viability_ratio'`. Wrapper's own label string is `'Financial Viability Analysis'`.
- **What it measures**: The header comment describes it as "Debt management and financial flexibility evaluation"; the report narrative states the ratio "measures the institution's ability to meet debt obligations from expendable net assets, indicating financial flexibility and debt management capacity." Values are read from bare four-digit year columns 2010–2024 in the source file; the analyzer does not itself compute the numerator or denominator.
- **Sector differences**:
  - `sectorName` is set from the sector code: `'1'` → Public, `'2'` → Private, `'3'` → For-Profit, anything else → Community College.
  - **Scoring branch**: sector `'1'` OR `'4'` uses percentile-relative bands against the peer group. All other sectors (including `'3'`) use the absolute private-university bands. The code comment justifying this reads: "Public: percentile-relative (absolute bands collapse under GASB 68/75)" and, on `viabilityStatusFromPercentile`, "GASB 68/75 drives public expendable net assets negative sector-wide, so the 0.10/0.20/0.30/0.40 bands degenerate into a sign test. Score relative position."
  - **Restriction-adjusted ratio (Tier 3A)**: computed and potentially substituted into scoring **only when `this.sector === '2'`** (Private), even though the doc comment on `scoreAdjustedViability` says the thresholds apply "to all sectors".
  - **`assessFinancialHealth`** (a reporting label only, not scored) branches on `sectorName === 'Public'`, not on the sector code. Public: `>= 0.40` Excellent, `>= 0.30` Good, `>= 0.20` Fair, `>= 0.10` Weak, else Critical. All other sector names (Private, For-Profit, Community College): `>= 0.50` Excellent, `>= 0.40` Good, `>= 0.30` Fair, `>= 0.20` Weak, else Critical.
  - Community college (`'4'`) is therefore scored on the *public* branch but labelled with the *private* health thresholds. This is what the code does; no reconciling comment exists.
- **Scoring**:

`maxPoints = this.maxRiskPoints = 8`. Derived component caps: `basePoints = Math.round(8 * 0.5) = 4`; `peerPoints = Math.round(8 * 0.25) = 2`; `currentPoints = Math.round(8 * 0.25) = 2`.

Factor 1 — 3-year rolling average (4 points max). Runs only if `rollingAverage != null && Number.isFinite(rollingAverage)`.

Public / Community College (sector `'1'` or `'4'`) — driven by `benchmarking.rollingPerformance.percentile` through `viabilityStatusFromPercentile`:

Percentile band (ceiling)	Status	Points
percentile < 10	critical	4
percentile < 25	weak	`Math.round(4 * 0.75)` = 3
percentile < 50	fair	`Math.round(4 * 0.5)` = 2
percentile < 75	good	`Math.round(4 * 0.25)` = 1
percentile >= 75	strong	0
percentile null / undefined / non-finite	unknown	0 (no branch taken)

All other sectors — absolute ceilings on the rolling average:

Rolling average (ceiling)	Points
< 0.20	4
< 0.30	3
< 0.40	2
< 0.50	1

Rolling average (ceiling)	Points
>= 0.50	0 (explicit empty `else`)

Factor 2 — current-year peer quartile (2 points max). Runs whenever `benchmarking.currentPerformance` exists. Applies to **all** sectors.

`currentPerformance.quartile`	Points
`'Bottom quartile'` (percentile < 25)	2
`'Below median'` (25 <= percentile < 50)	`Math.round(2 * 0.5)` = 1
`'Above median'` or `'Top quartile'`	0 (explicit empty `else`)

Factor 3 — current-year distress (2 points max). Runs if `currentRatio !== null`.

Public / Community College (`'1'` or `'4'`) — on `benchmarking.currentPerformance.percentile`:

Percentile band (ceiling)	Status	Points
percentile < 10	critical	2
percentile < 25	weak	`Math.round(2 * 0.67)` = 1
percentile < 50	fair	`Math.round(2 * 0.33)` = 1
percentile >= 50	good / strong	0

All other sectors — absolute ceilings on the current-year ratio:

Current ratio (ceiling)	Points
< 0.10	2
< 0.20	`Math.round(2 * 0.67)` = 1
< 0.30	`Math.round(2 * 0.33)` = 1
>= 0.30	0

Total is then `Math.min(riskPoints, 8)`.

Alternate ladder — `scoreAdjustedViability` (restriction-adjusted, Private only). Replaces the whole standard score when `adjustedAssessment.totalRiskPoints > riskAssessment.totalRiskPoints` ("Use adjusted scoring when it's stricter than the standard scoring"). Adjusted ratio = unrestricted net assets ÷ long-term debt.

Adjusted ratio (ceiling)	Points	riskLevel assigned directly
< 0.5	8	Critical
< 1.0	`Math.round(8 * 0.75)` = 6	High
< 1.5	`Math.round(8 * 0.5)` = 4	Moderate
< 2.0	`Math.round(8 * 0.25)` = 2	Low-Moderate
>= 2.0	0	Low

- **Maximum points**: `this.maxRiskPoints = 8`, set in the constructor and returned as `maxRiskPoints` on every result. The HTML wrapper unconditionally overwrites `result.analysisResults.maxRiskPoints = 8` with the comment "Enhanced framework: 3+2+3 points" — a 3+2+3 decomposition that does not match the 4+2+2 the analyzer actually computes.
- **Risk level mapping**: In the standard path, from raw points against `maxPoints = 8`:

Condition	riskLevel
riskPoints >= 8 × 0.75 = 6	High
riskPoints >= 8 × 0.5 = 4	Moderate
riskPoints >= 8 × 0.25 = 2	Low-Moderate

Condition	riskLevel
otherwise	Low

All are floors. The standard path never produces `'Critical'`; that string can only arrive via `scoreAdjustedViability`, which sets `riskLevel` directly from the adjusted ratio band (table above) rather than from the points percentage. An unused `determineRiskLevel(points)` helper also exists using the same 75 / 50 / 25 percentage floors and returning High / Moderate / Low-Moderate / Low.

- **Data-gap behaviour**: `createErrorResult` returns `success: false`, `analysisResults: { riskPoints: 0, maxRiskPoints: 8, riskLevel: 'Unknown', isDataUnavailable: true }`. Triggered by: missing `unitId`; institution absent from the viability dataset; no usable current viability ratio ("Insufficient viability data"); and any thrown exception. UNCLEAR: `createErrorResult(unitId, institutionName, errorMessage)` takes three parameters but two of its four call sites pass only two arguments (`'MISSING_UNITID'` / `'ANALYSIS_FAILED'` plus a message), so in those cases the message lands in `institutionName` and `errorMessage` is `undefined`. Separately, the HTML wrapper returns `riskPoints: 5, maxRiskPoints: 8, riskLevel: 'Critical'` (no `isDataUnavailable`) when the institution object has no `unitid`, and `riskPoints: 0, maxRiskPoints: 8, riskLevel: 'Unknown', isDataUnavailable: true` when the viability data file is not loaded.
- **Peer logic**: `this.institutions` is built by `extractAllInstitutions()`, which keeps only rows whose SECTOR column equals `this.sector` (falling back to `this.sector` if no sector column is found). `findAndAnalyzePeerGroup` then excludes self by UNITID and applies, when supplied: Carnegie basic (`peerCriteria.carnegieBasic`, string-compared), HBCU (`'hbcu-only'` keeps `hbcu === '1'`; `'non-hbcu-only'` drops it), and an explicit `customPeerGroup` UNITID list. `dataQuality` is `'Good'` at >= 5 peers, else `'Limited'`. The comparison used for scoring is a **percentile rank**: `benchmarkValue` counts peers strictly below the institution's value (`institutionValue > v`, higher-is-better) and reports `Math.round(count / peers * 100)`, then maps `>= 75` Top quartile/Excellent, `>= 50` Above median/Good, `>= 25` Below median/Below Average, else Bottom quartile/Poor. `calculateStatistics` computes both mean and median; the **median** is what the report text and `generatePeerComparison` use — although `generatePeerComparison` reads `peerStatistics.median` and then prints it labelled "peer average".
- **Notes**:
  - `static TREND_START_YEAR = 2019`. Comment: "Endpoint comparison across 2010-2024 straddles GASB 68 (FY2015) and GASB 75 (FY2018); both produce one-time step-downs in expendable net assets that a first-vs-last comparison reads as a sustained multi-year decline. 2019 is the first year clean of both. Keep in sync with the primary reserve analyzer or the two measures report trend on different bases." Trend direction is Improving if totalChange > 0.05, Declining if < -0.05, else Stable; `'Unknown'` when fewer than two observations fall in the window.
  - **2026-08-28 (v49.1)** — Factor 1 guard widened from `!== null` to `!= null`. Quoting the code: "WAS WRONG: when calculateRollingAverages could not build a recent 3-year average it left recent3Year undefined, which passes `!== null`. Every threshold comparison (`undefined < 0.20` etc.) is then false, so the institution fell through all four bands and silently scored 0 on this 4-point factor — a data gap scored as a clean result. EVIDENCE: exactly 6 institutions reach that path. NOW: a missing rolling average SKIPS the factor and is reported as a data gap in the risk-factor text; it is never scored as zero risk."
  - **2026-08-28 (v49.1)** — `rollingAverages.recent3Year` is now explicitly initialised to `null`. Comment: "undefined passes a `!== null` guard, so it flowed through the Factor 1 scorer and the report generator and then threw on `.toFixed(3)` in generateDetailedReport. EVIDENCE: exactly 6 institutions reach that path... NOW: an absent rolling average is null, distinguishable from a real value; callers use `!= null` and skip the factor rather than award zero."
  - **2026-08-28 (v49.1)** — a redundant `rollingAverage !== null` clause was removed from the Factor 2 guard. Comment: "Factor 2 scores the CURRENT-year peer quartile and never reads rollingAverage... now that the absent case is a real null, leaving the clause in place would newly suppress Factor 2 for those institutions — a score change that is not authorised here. Behaviour is therefore unchanged for every institution."
  - **2026-08-28 (v49.1)** — the report generator's rolling-average guard was widened the same way, and an explicit "3-year rolling average: not available (fewer than two usable observations in the most recent three years)" finding is now emitted instead of silent omission.
  - **2026-08-26** — a UNITID→row `Map` index (`__unitRow`) replaced a linear scan. The comment states it is "pure performance", is deliberately FIRST-WINS on duplicate UNITIDs because "A Map built forward with an unconditional set() keeps the LAST duplicate instead, which would silently change which row is analysed...

That would be a scoring change", and quantifies the prior cost as "roughly 624,000 row comparisons per institution per analyzer - about 7.4 billion for a full ranking run, which is why ranking fell over while single-institution analysis was fine."

- `computeRestrictionAdjustedRatio` searches back up to 5 years from the latest viability year for a year where both the UNA file (bare `20\d{2}` header) and the debt file (`20\d{2} d` header, deliberately excluding `20\d{2} dr`) carry values; long-term debt must be `> 0`. When it falls back to an earlier year, `yearFallback` is set and a note appears in the report.
- Factor 3's public branch awards the same 1 point for the "weak" and "fair" bands (`Math.round(2 * 0.67)` and `Math.round(2 * 0.33)` both round to 1), as does the private branch for the `<0.20` and `<0.30` bands.
- `calculatePeerRanking` and `calculateQuartile` are defined but not called anywhere in this file.
