Case Study 3

AI Scoring Engine

"From Evidence to Scores"

Labs 5-6 — Weeks 5-6 — **Enhanced v2**

Course: Big Data and Intelligent Analytics
Term: Spring 2026

Designed by

Sri Krishnamurthy, CFA

sri@quantuniversity.com

Assigned: February 6, 2026
Due: February 20, 2026 (3:59 PM)

You have the platform (CS1). You have the evidence (CS2). Now score it.

This enhanced case study builds the **complete scoring pipeline**—from CS2 evidence through **dimension mapping**, **rubric scoring**, and **full Org-AI-R calculation** with **property-based testing**.

NEW: Evidence mapping, Glassdoor/Board data collection, Integration service

Weight	8% of final grade
Format	Individual work
Prerequisites	**Working CS1** + **Working CS2**
Companies	NVDA, JPM, WMT, GE, DG (5 real companies)
Testing	Property-based tests with Hypothesis required
NEW Tasks	7 new tasks connecting CS2 → CS3

The Complete Org-AI-R Formula

$$\text{Org-AI-R}_{j,t} = (1-\beta)\cdot\left[\alpha\cdot V^R_{org,j}(t) + (1-\alpha)\cdot H^R_{org,k}(t)\right] + \beta\cdot\text{Synergy}(V^R, H^R)$$

$\alpha = 0.60$	Idiosyncratic weight	(company-specific factors)
$\beta = 0.12$	Synergy weight	(alignment effects)
$\lambda = 0.25$	Non-compensatory penalty	(balance requirement)
$\delta = 0.15$	Position adjustment	**(CORRECTED in v3.0)**

Contents

Part I

Lab 5: Evidence to V^R

1 Lab 5 Preamble

Date	February 6, 2026 (Thursday)
Focus	Evidence mapping, Rubric scoring, V^R calculation, Property-based testing
Time Estimate	Lecture: 2 hrs — Lab: 6 hrs — Challenge: +3 hrs

1.1 Learning Objectives

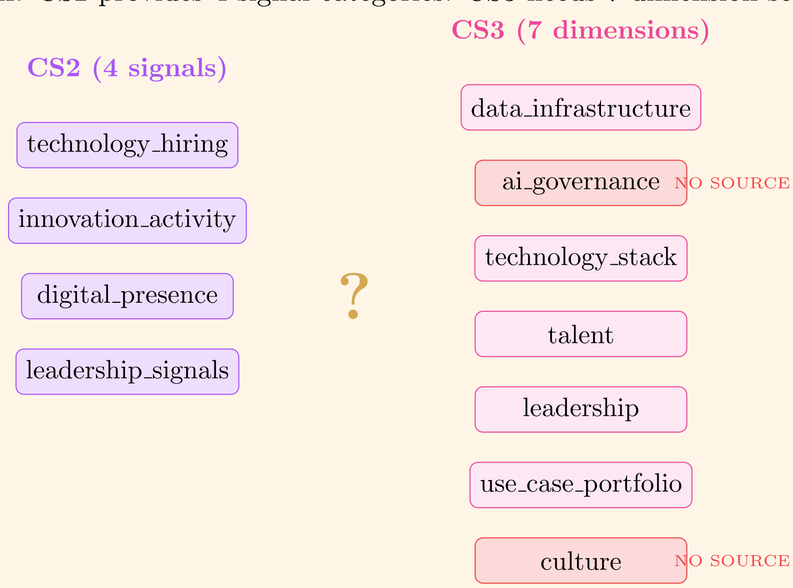
Learning Objectives (Bloom's Taxonomy)

Bloom's Level	Objective
Remember	State the V^R formula and 7 dimension names
Understand	Explain how CS2 signals map to V^R dimensions
Apply	Implement evidence-to-dimension mapping with weights
Analyze	Compare property-based vs example-based testing
Evaluate	Assess rubric scoring accuracy against expected ranges
Create	Design new data collectors for Governance and Culture gaps

1.2 The CS2 → CS3 Gap

Warning

The Problem: CS2 provides 4 signal categories. CS3 needs 7 dimension scores.



This case study fills the gap with evidence mapping, new data collectors, and rubric scoring.

2 Task 5.0a: Evidence-to-Dimension Mapper [NEW]

NEW:

Why This Task Matters CS2 collects evidence in 4 signal categories. V^R scoring requires 7 dimension scores. This mapper defines the explicit relationship between signals and dimensions with contribution weights.

2.1 The Mapping Table

Each CS2 signal contributes to multiple dimensions with different weights:

Table 1: Signal-to-Dimension Mapping Matrix

CS2 Source	Data	Gov	Tech	Talent	Lead	Use	Culture
technology_hiring	0.10	—	0.20	**0.70**	—	—	0.10
innovation_activity	0.20	—	**0.50**	—	—	0.30	—
digital_presence	**0.60**	—	0.40	—	—	—	—
leadership_signals	—	0.25	—	—	**0.60**	—	0.15
sec_item_1 (Business)	—	—	0.30	—	—	**0.70**	—
sec_item_1a (Risk)	0.20	**0.80**	—	—	—	—	—
sec_item_7 (MD&A)	0.20	—	—	—	**0.50**	0.30	—
glassdoor_reviews [NEW]	—	—	—	0.10	0.10	—	**0.80**
board_composition [NEW]	—	**0.70**	—	—	0.30	—	—

Bold = Primary contribution. Weights within a source sum to 1.0.

2.2 Implementation

scoring/evidence_mapper.py — Data Models

```
from dataclasses import dataclass, field
from typing import Dict, List, Optional
from enum import Enum
from decimal import Decimal

class Dimension(str, Enum):
    DATA_INFRASTRUCTURE = "data_infrastructure"
    AI_GOVERNANCE = "ai_governance"
    TECHNOLOGY_STACK = "technology_stack"
    TALENT = "talent"
    LEADERSHIP = "leadership"
    USE_CASE_PORTFOLIO = "use_case_portfolio"
    CULTURE = "culture"

class SignalSource(str, Enum):
    # CS2 External Signals
    TECHNOLOGY_HIRING = "technology_hiring"
    INNOVATION_ACTIVITY = "innovation_activity"
    DIGITAL_PRESENCE = "digital_presence"
    LEADERSHIP_SIGNALS = "leadership_signals"
    # CS2 SEC Sections
    SEC_ITEM_1 = "sec_item_1"
    SEC_ITEM_1A = "sec_item_1a"
    SEC_ITEM_7 = "sec_item_7"
```



```python
    # CS3 New Sources
    GLASSDOOR_REVIEWS = "glassdoor_reviews"
    BOARD_COMPOSITION = "board_composition"

@dataclass
class DimensionMapping:
    """Maps a signal source to dimensions with weights."""
    source: SignalSource
    primary_dimension: Dimension
    primary_weight: Decimal
    secondary_mappings: Dict[Dimension, Decimal] = field(default_factory=dict)
    reliability: Decimal = Decimal("0.8")  # Source reliability

@dataclass
class EvidenceScore:
    """A score from a single evidence source."""
    source: SignalSource
    raw_score: Decimal        # 0-100
    confidence: Decimal       # 0-1
    evidence_count: int
    metadata: Dict = field(default_factory=dict)

@dataclass
class DimensionScore:
    """Aggregated score for one dimension."""
    dimension: Dimension
    score: Decimal
    contributing_sources: List[SignalSource]
    total_weight: Decimal
    confidence: Decimal
```

scoring/evidence_mapper.py — YOUR IMPLEMENTATION

```python
# THE CRITICAL MAPPING TABLE
SIGNAL_TO_DIMENSION_MAP: Dict[SignalSource, DimensionMapping] = {
    SignalSource.TECHNOLOGY_HIRING: DimensionMapping(
        source=SignalSource.TECHNOLOGY_HIRING,
        primary_dimension=Dimension.TALENT,
        primary_weight=Decimal("0.70"),
        secondary_mappings={
            Dimension.TECHNOLOGY_STACK: Decimal("0.20"),
            Dimension.CULTURE: Decimal("0.10"),
        },
        reliability=Decimal("0.85"),
    ),
    SignalSource.INNOVATION_ACTIVITY: DimensionMapping(
        source=SignalSource.INNOVATION_ACTIVITY,
        primary_dimension=Dimension.TECHNOLOGY_STACK,
        primary_weight=Decimal("0.50"),
        secondary_mappings={
            Dimension.USE_CASE_PORTFOLIO: Decimal("0.30"),
            Dimension.DATA_INFRASTRUCTURE: Decimal("0.20"),
        },
        reliability=Decimal("0.80"),
    ),
    # TODO: Add remaining mappings for:
    # - DIGITAL_PRESENCE → DATA_INFRASTRUCTURE (primary)
    # - LEADERSHIP_SIGNALS → LEADERSHIP (primary)
    # - SEC_ITEM_1 → USE_CASE_PORTFOLIO (primary)
    # - SEC_ITEM_1A → AI_GOVERNANCE (primary)
    # - SEC_ITEM_7 → LEADERSHIP (primary)
    # - GLASSDOOR_REVIEWS → CULTURE (primary)
    # - BOARD_COMPOSITION → AI_GOVERNANCE (primary)
}

class EvidenceMapper:
    """Maps CS2 evidence to 7 V^R dimensions."""
```

```python

    def __init__(self):
        self.mappings = SIGNAL_TO_DIMENSION_MAP

    def map_evidence_to_dimensions(
        self,
        evidence_scores: List[EvidenceScore],
    ) -> Dict[Dimension, DimensionScore]:
        """
        Convert CS2 evidence scores to 7 dimension scores.

        Algorithm:
        1. Initialize accumulators for each dimension
        2. For each evidence source:
           a. Look up its mapping
           b. Add weighted contribution to primary dimension
           c. Add weighted contributions to secondary dimensions
        3. Calculate weighted average for each dimension
        4. Return DimensionScore for all 7 dimensions

        IMPORTANT: Dimensions with NO evidence should default to 50.0

        Args:
            evidence_scores: List of scores from CS2 + CS3 sources

        Returns:
            Dict mapping each Dimension to its aggregated score
        """
        # TODO: Initialize dimension accumulators
        # dimension_sums: Dict[Dimension, Decimal] = {d: Decimal(0) for d in Dimension}
        # dimension_weights: Dict[Dimension, Decimal] = {d: Decimal(0) for d in Dimension}
        # dimension_sources: Dict[Dimension, List[SignalSource]] = {d: [] for d in Dimension}

        # TODO: Process each evidence score
        # for ev in evidence_scores:
        #     mapping = self.mappings.get(ev.source)
        #     if not mapping:
        #         continue
        #
        #     # Weight by confidence and reliability
        #     effective_score = ev.raw_score * ev.confidence * mapping.reliability
        #
        #     # Primary contribution
        #     dim = mapping.primary_dimension
        #     weight = mapping.primary_weight
        #     dimension_sums[dim] += effective_score * weight
        #     dimension_weights[dim] += weight * ev.confidence * mapping.reliability
        #     dimension_sources[dim].append(ev.source)
        #
        #     # Secondary contributions
        #     for dim, weight in mapping.secondary_mappings.items():
        #         ... (same pattern)

        # TODO: Calculate weighted averages
        # TODO: Default to 50.0 for dimensions with no evidence
        # TODO: Return Dict[Dimension, DimensionScore]

        raise NotImplementedError("Implement map_evidence_to_dimensions")

    def get_coverage_report(
        self,
        evidence_scores: List[EvidenceScore],
    ) -> Dict[Dimension, Dict]:
        """
        Report which dimensions have evidence and which have gaps.

        Returns dict with:
        - has_evidence: bool
```



```
        - source_count: int
        - total_weight: float
        - confidence: float
        """
        # TODO: Implement coverage analysis
        raise NotImplementedError("Implement get_coverage_report")
```

Hint

Testing Your Mapper:

Property tests to write:

1. All 7 dimensions are always returned
2. All scores are bounded [0, 100]
3. Dimensions with no evidence default to 50
4. Adding more evidence sources doesn't decrease confidence

3 Task 5.0b: Rubric-Based Scorer [NEW]

NEW:

Why This Task Matters Raw scores (e.g., "42 AI jobs") need interpretation. The PE Org-AI-R framework defines 5-level rubrics for each dimension. This scorer converts evidence into rubric-aligned scores.

3.1 The 7 Dimension Rubrics

Scoring Rubric:

Data Infrastructure

Lvl	Range	Criteria	Keywords
5	80-100	Modern cloud platform (Snowflake, Databricks), data quality >90%, real-time pipelines, API-first architecture	snowflake, databricks, lakehouse, real-time
4	60-79	Hybrid cloud environment, data quality 70-90%, batch pipelines, partial data catalog	azure, aws, warehouse, etl
3	40-59	Legacy with modernization roadmap, quality 50-70%, cloud adoption in progress	migration, hybrid, modernizing
2	20-39	On-premise legacy systems, quality <50%, siloed data stores	legacy, silos, on-premise
1	0-19	No modern infrastructure, fragmented data, manual processes	mainframe, spreadsheets, manual

Scoring Rubric:

AI Governance

Lvl	Range	Criteria	Keywords
5	80-100	CAIO/CDO reports to CEO, board AI committee, comprehensive model risk management framework	caio, cdo, board committee, model risk
4	60-79	VP-level AI sponsor, documented AI policies, risk assessment process in place	vp data, ai policy, risk framework
3	40-59	Director-level ownership, basic policies exist, IT-led governance structure	director, guidelines, it governance
2	20-39	Informal governance only, no documented policies, ad-hoc oversight	informal, no policy, ad-hoc
1	0-19	No governance structure, no AI oversight, unmanaged risk exposure	none, no oversight, unmanaged

Scoring Rubric:

Talent

Lvl	Range	Criteria	Keywords
5	80-100	Large AI/ML team (>20 specialists), <10% turnover, internal ML platform team, research capability	ml platform, ai research, large team
4	60-79	Established team (10-20 professionals), active hiring pipeline, retention programs	data science team, ml engineers
3	40-59	Small team (3-10 data scientists), growing capability, some turnover challenges	data scientist, growing team
2	20-39	1-2 data scientists, high turnover rate, limited technical depth	junior, contractor, turnover
1	0-19	No dedicated AI/ML talent, dependent on vendors/contractors	no data scientist, vendor only

Scoring Rubric:

Technology Stack

Lvl	Range	Criteria	Keywords
5	80-100	Full MLOps platform (SageMaker, Vertex AI), feature store, model registry, automated pipelines	sagemaker, mlops, feature store
4	60-79	ML platform adopted (Databricks ML, MLflow), experiment tracking, partial automation	mlflow, kubeflow, databricks ml
3	40-59	Basic ML tools in use, manual deployment, notebook-based development	jupyter, notebooks, manual deploy
2	20-39	Spreadsheet-based analytics only, no ML tooling, basic BI tools	excel, tableau only, no ml
1	0-19	No analytics capability, manual reporting processes	manual, no tools

Scoring Rubric:

Leadership

Lvl	Range	Criteria	Keywords
5	80-100	CEO publicly champions AI, board AI/tech committee, documented multi-year AI strategic plan	ceo ai, board committee, ai strategy
4	60-79	C-suite sponsor (CTO/CDO), AI in strategy documents, executive engagement	cto ai, strategic priority
3	40-59	VP-level sponsorship, departmental AI initiatives underway	vp sponsor, department initiative
2	20-39	Limited executive awareness, IT-driven initiatives only	it led, limited awareness
1	0-19	No executive sponsorship, AI not discussed at leadership level	no sponsor, not discussed

Scoring Rubric:

Use Case Portfolio

Lvl	Range	Criteria	Keywords
5	80-100	5+ AI use cases in production, documented ROI of 3x+, revenue-generating AI products	production ai, 3x roi, ai product
4	60-79	2-4 use cases in production, measured positive ROI, scaling plans in progress	production, measured roi, scaling
3	40-59	1-2 pilots moved to production, early ROI tracking underway	pilot, early production
2	20-39	POCs/proofs of concept only, no production deployments yet	poc, proof of concept
1	0-19	No AI use cases, exploration phase only	exploring, no use cases

Scoring Rubric:

Culture

Lvl	Range	Criteria	Keywords
5	80-100	Innovation celebrated and rewarded, fail-fast experimentation culture, data-driven decisions embedded	innovative, data-driven, fail-fast
4	60-79	Experimentation encouraged, growing data literacy across teams	experimental, learning culture
3	40-59	Open to change, some resistance from middle management, mixed data adoption	open to change, some resistance
2	20-39	Change resistant culture, hierarchical decision making, intuition over data	bureaucratic, resistant, slow
1	0-19	Hostile to change, siloed organization, no data culture	hostile, siloed, no data culture

3.2 Implementation

scoring/rubric_scorer.py — YOUR IMPLEMENTATION

```python
from dataclasses import dataclass
from typing import Dict, List, Tuple
from enum import Enum
from decimal import Decimal
import re

class ScoreLevel(Enum):
    LEVEL_5 = (80, 100, "Excellent")
    LEVEL_4 = (60, 79, "Good")
    LEVEL_3 = (40, 59, "Adequate")
    LEVEL_2 = (20, 39, "Developing")
    LEVEL_1 = (0, 19, "Nascent")

    @property
    def min_score(self) -> int:
        return self.value[0]

    @property
    def max_score(self) -> int:
        return self.value[1]

@dataclass
class RubricCriteria:
    """Criteria for a single rubric level."""
    level: ScoreLevel
```

```python
    keywords: List[str]
    min_keyword_matches: int
    quantitative_threshold: float  # e.g., AI job ratio > 0.3

@dataclass
class RubricResult:
    """Result of rubric scoring."""
    dimension: str
    level: ScoreLevel
    score: Decimal
    matched_keywords: List[str]
    keyword_match_count: int
    confidence: Decimal
    rationale: str

# Rubric definitions - YOU COMPLETE THIS
DIMENSION_RUBRICS: Dict[str, Dict[ScoreLevel, RubricCriteria]] = {
    "talent": {
        ScoreLevel.LEVEL_5: RubricCriteria(
            level=ScoreLevel.LEVEL_5,
            keywords=["ml platform", "ai research", "large team", ">20 specialists",
                      "ai leadership", "principal ml", "staff ml"],
            min_keyword_matches=3,
            quantitative_threshold=0.40,  # >40% AI job ratio
        ),
        ScoreLevel.LEVEL_4: RubricCriteria(
            level=ScoreLevel.LEVEL_4,
            keywords=["data science team", "ml engineers", "10-20",
                      "active hiring", "retention"],
            min_keyword_matches=2,
            quantitative_threshold=0.25,
        ),
        # TODO: Add LEVEL_3, LEVEL_2, LEVEL_1
    },
    # TODO: Add rubrics for all 7 dimensions
}

class RubricScorer:
    """Score evidence against PE Org-AI-R rubrics."""

    def __init__(self):
        self.rubrics = DIMENSION_RUBRICS

    def score_dimension(
        self,
        dimension: str,
        evidence_text: str,
        quantitative_metrics: Dict[str, float],
    ) -> RubricResult:
        """
        Score a dimension using rubric matching.

        Algorithm:
        1. Normalize evidence text (lowercase)
        2. For each level (5 down to 1):
           a. Count keyword matches
           b. Check quantitative threshold
           c. If criteria met, return score in that level's range
        3. Use keyword density to interpolate within range

        Args:
            dimension: One of the 7 dimension names
            evidence_text: Concatenated evidence text
            quantitative_metrics: Dict of metric_name -> value
                e.g., {"ai_job_ratio": 0.35, "patent_count": 12}

        Returns:
            RubricResult with score, level, and matched keywords
```



```
        """
        # TODO: Normalize text
        # text = evidence_text.lower()

        # TODO: Get rubric for dimension
        # rubric = self.rubrics.get(dimension, {})

        # TODO: Check each level from 5 to 1
        # for level in [ScoreLevel.LEVEL_5, LEVEL_4, ...]:
        #     criteria = rubric.get(level)
        #     if not criteria:
        #         continue
        #
        #     # Count keyword matches
        #     matches = [kw for kw in criteria.keywords if kw in text]
        #
        #     # Check if criteria met
        #     if len(matches) >= criteria.min_keyword_matches:
        #         # Interpolate score within level range
        #         # More matches = higher in range
        #         ...

        raise NotImplementedError("Implement score_dimension")

    def score_all_dimensions(
        self,
        evidence_by_dimension: Dict[str, str],
        metrics_by_dimension: Dict[str, Dict[str, float]],
    ) → Dict[str, RubricResult]:
        """Score all 7 dimensions."""
        # TODO: Call score_dimension for each dimension
        raise NotImplementedError("Implement score_all_dimensions")
```

4 Task 5.0c: Glassdoor Culture Collector [NEW]

NEW:

Filling the Culture Gap CS2 has **no evidence source** for the Culture dimension. This task adds a Glassdoor review analyzer that extracts culture signals from employee reviews.

4.1 Culture Signal Categories

Table 2: Glassdoor Keywords for Culture Scoring

Category	Keywords
Innovation (Positive)	innovative, cutting-edge, forward-thinking, encourages new ideas, experimental, creative freedom, startup mentality
Innovation (Negative)	bureaucratic, slow to change, resistant, outdated, stuck in old ways, red tape
Data-Driven	data-driven, metrics, evidence-based, analytical, KPIs, dashboards, data culture
AI Awareness	AI, machine learning, automation, data science, ML, algorithms, artificial intelligence
Change Readiness	agile, adaptive, fast-paced, embraces change, continuous improvement
Change Resistance	rigid, traditional, slow, risk-averse, change resistant

4.2 Implementation

pipelines/glassdoor_collector.py — Data Models

```python
from dataclasses import dataclass, field
from typing import List, Optional
from datetime import datetime
from decimal import Decimal

@dataclass
class GlassdoorReview:
    """A single Glassdoor review."""
    review_id: str
    rating: float           # 1-5 stars
    title: str
    pros: str
    cons: str
    advice_to_management: Optional[str]
    is_current_employee: bool
    job_title: str
    review_date: datetime

@dataclass
class CultureSignal:
    """Aggregated culture signal from Glassdoor."""
    company_id: str
    ticker: str

```

```python
    # Component scores (0-100)
    innovation_score: Decimal
    data_driven_score: Decimal
    change_readiness_score: Decimal
    ai_awareness_score: Decimal

    # Aggregate
    overall_score: Decimal

    # Metadata
    review_count: int
    avg_rating: Decimal
    current_employee_ratio: Decimal
    confidence: Decimal

    # Evidence
    positive_keywords_found: List[str] = field(default_factory=list)
    negative_keywords_found: List[str] = field(default_factory=list)
```

pipelines/glassdoor_collector.py — YOUR IMPLEMENTATION

```python
class GlassdoorCultureCollector:
    """
    Collect and analyze Glassdoor reviews for culture signals.

    Scoring Formula:
    - innovation_score = (positive_mentions - negative_mentions) / total_reviews * 50 + 50
    - data_driven_score = data_mentions / total_reviews * 100
    - ai_awareness_score = ai_mentions / total_reviews * 100
    - change_readiness = (positive_change - negative_change) / total * 50 + 50

    Overall = 0.30 * innovation + 0.25 * data_driven + 0.25 * ai_awareness + 0.20 * change
    """

    INNOVATION_POSITIVE = [
        "innovative", "cutting-edge", "forward-thinking",
        "encourages new ideas", "experimental", "creative freedom",
        "startup mentality", "move fast", "disruptive"
    ]

    INNOVATION_NEGATIVE = [
        "bureaucratic", "slow to change", "resistant",
        "outdated", "stuck in old ways", "red tape",
        "politics", "siloed", "hierarchical"
    ]

    DATA_DRIVEN_KEYWORDS = [
        "data-driven", "metrics", "evidence-based",
        "analytical", "kpis", "dashboards", "data culture",
        "measurement", "quantitative"
    ]

    AI_AWARENESS_KEYWORDS = [
        "ai", "artificial intelligence", "machine learning",
        "automation", "data science", "ml", "algorithms",
        "predictive", "neural network"
    ]

    CHANGE_POSITIVE = [
        "agile", "adaptive", "fast-paced", "embraces change",
        "continuous improvement", "growth mindset"
    ]

    CHANGE_NEGATIVE = [
        "rigid", "traditional", "slow", "risk-averse",
        "change resistant", "old school"
    ]
```



```python

    def analyze_reviews(
        self,
        company_id: str,
        ticker: str,
        reviews: List[GlassdoorReview],
    ) -> CultureSignal:
        """
        Analyze reviews for culture indicators.

        Algorithm:
        1. Combine pros and cons text for each review
        2. Count keyword matches for each category
        3. Weight by recency (last 2 years = full weight, older = 0.5)
        4. Weight current employees higher (1.2x multiplier)
        5. Calculate component scores
        6. Calculate overall weighted average

        Args:
            company_id: Company UUID
            ticker: Stock ticker
            reviews: List of Glassdoor reviews

        Returns:
            CultureSignal with all component scores
        """
        # TODO: Initialize counters
        # innovation_positive = 0
        # innovation_negative = 0
        # data_driven_mentions = 0
        # ai_awareness_mentions = 0
        # change_positive = 0
        # change_negative = 0
        # total_weight = 0

        # TODO: Process each review
        # for review in reviews:
        #     text = f"{review.pros} {review.cons}".lower()
        #
        #     # Calculate recency weight
        #     days_old = (datetime.now() - review.review_date).days
        #     recency_weight = 1.0 if days_old < 730 else 0.5
        #
        #     # Calculate employee weight
        #     employee_weight = 1.2 if review.is_current_employee else 1.0
        #
        #     weight = recency_weight * employee_weight
        #     total_weight += weight
        #
        #     # Count keywords (weighted)
        #     for kw in self.INNOVATION_POSITIVE:
        #         if kw in text:
        #             innovation_positive += weight
        #     # ... repeat for other categories

        # TODO: Calculate scores (bounded 0-100)
        # TODO: Calculate confidence based on review count
        # TODO: Return CultureSignal

        raise NotImplementedError("Implement analyze_reviews")

    def fetch_reviews(self, ticker: str, limit: int = 100) -> List[GlassdoorReview]:
        """
        Fetch reviews from Glassdoor (or cached data).

        NOTE: In production, use Glassdoor API or web scraping.
        For this lab, use the provided sample data files.
        """
```

```
        # TODO: Load from data/glassdoor/{ticker}.json
        raise NotImplementedError("Implement fetch_reviews")
```

5 Task 5.0d: Board Composition Analyzer [NEW]

NEW:

Filling the AI Governance Gap CS2 has **limited evidence** for the AI Governance dimension. SEC filings mention risk factors, but don't capture board-level AI oversight. This task analyzes board composition from proxy statements.

5.1 Governance Indicators from Board Data

Table 3: Board Composition Scoring Criteria

Indicator	Points	Source
Technology/Digital committee exists	+15	Proxy statement
Board member with AI/ML expertise	+20	Director bios
CAIO, CDO, or CTO on executive team	+15	Executive team page
Independent director ratio > 0.5	+10	Proxy statement
Risk committee with tech oversight	+10	Committee charters
AI mentioned in strategic priorities	+10	Annual report
Base score	20	
Maximum	100	

5.2 Implementation

pipelines/board_analyzer.py — Data Models

```python
from dataclasses import dataclass, field
from typing import List, Optional
from decimal import Decimal

@dataclass
class BoardMember:
    """A board member or executive."""
    name: str
    title: str
    committees: List[str]
    bio: str  # Background text
    is_independent: bool
    tenure_years: int

@dataclass
class GovernanceSignal:
    """Board-derived governance signal."""
    company_id: str
    ticker: str

    # Boolean indicators
    has_tech_committee: bool
    has_ai_expertise: bool
    has_data_officer: bool
    has_risk_tech_oversight: bool
    has_ai_in_strategy: bool

    # Metrics
    tech_expertise_count: int
```

```python
    independent_ratio: Decimal

    # Final score
    governance_score: Decimal
    confidence: Decimal

    # Evidence
    ai_experts: List[str] = field(default_factory=list)  # Names
    relevant_committees: List[str] = field(default_factory=list)
```

pipelines/board_analyzer.py — YOUR IMPLEMENTATION

```python
class BoardCompositionAnalyzer:
    """
    Analyze board composition for AI governance indicators.

    Scoring:
    - Tech committee exists: +15 points
    - AI expertise on board: +20 points
    - Data officer role: +15 points
    - Independent ratio > 0.5: +10 points
    - Risk committee tech oversight: +10 points
    - AI in strategic priorities: +10 points
    - Base: 20 points
    - Max: 100 points
    """

    AI_EXPERTISE_KEYWORDS = [
        "artificial intelligence", "machine learning",
        "chief data officer", "cdo", "caio", "chief ai",
        "chief technology", "cto", "chief digital",
        "data science", "analytics", "digital transformation"
    ]

    TECH_COMMITTEE_NAMES = [
        "technology committee", "digital committee",
        "innovation committee", "it committee",
        "technology and cybersecurity"
    ]

    DATA_OFFICER_TITLES = [
        "chief data officer", "cdo",
        "chief ai officer", "caio",
        "chief analytics officer", "cao",
        "chief digital officer"
    ]

    def analyze_board(
        self,
        company_id: str,
        ticker: str,
        members: List[BoardMember],
        committees: List[str],
        strategy_text: str = "",
    ) -> GovernanceSignal:
        """
        Analyze board for AI governance strength.

        Args:
            company_id: Company UUID
            ticker: Stock ticker
            members: List of board members and executives
            committees: List of committee names
            strategy_text: Text from annual report strategy section

        Returns:
            GovernanceSignal with governance score
```



```
        """
        score = Decimal("20")  # Base score

        # TODO: Check for tech committee
        # has_tech = any(
        #     any(tc in c.lower() for tc in self.TECH_COMMITTEE_NAMES)
        #     for c in committees
        # )
        # if has_tech:
        #     score += Decimal("15")

        # TODO: Check for AI expertise on board
        # ai_experts = []
        # for member in members:
        #     bio_lower = member.bio.lower()
        #     if any(kw in bio_lower for kw in self.AI_EXPERTISE_KEYWORDS):
        #         ai_experts.append(member.name)
        # if ai_experts:
        #     score += Decimal("20")

        # TODO: Check for data officer role
        # TODO: Check independent ratio
        # TODO: Check risk committee oversight
        # TODO: Check AI in strategy

        # TODO: Cap at 100
        # score = min(score, Decimal("100"))

        # TODO: Calculate confidence
        # confidence = min(Decimal("0.5") + len(members) / 20, Decimal("0.95"))

        raise NotImplementedError("Implement analyze_board")

    def extract_from_proxy(self, proxy_html: str) → tuple:
        """
        Extract board members and committees from DEF 14A proxy statement.

        Returns:
            (List[BoardMember], List[str] committees)
        """
        # TODO: Parse proxy statement HTML
        # This is complex - use BeautifulSoup + regex patterns
        raise NotImplementedError("Implement extract_from_proxy")
```

6 Task 5.0e: Talent Concentration Calculator [NEW]

NEW:

What is Talent Concentration? Talent Concentration (TC) measures **key-person risk** — how much AI capability depends on a few individuals.

- TC = 0.0: Capability distributed across many people (low risk)
- TC = 1.0: All capability in one person (maximum risk)

The TalentRiskAdj formula penalizes high TC: TalentRiskAdj $= 1 - 0.15 \times \max(0, TC - 0.25)$

6.1 TC Calculation from Job Data

Table 4: Talent Concentration Indicators

Indicator	Interpretation	TC Effect
High leadership ratio	AI capability concentrated in few leaders	↑ TC
Small team size	Fewer people = higher concentration	↑ TC
Mentions of individuals	Glassdoor mentions specific people	↑ TC
Diverse skill requirements	Multiple specializations needed	↓ TC
Active junior hiring	Building distributed capability	↓ TC

6.2 Implementation

scoring/talent_concentration.py — YOUR IMPLEMENTATION

```python
from dataclasses import dataclass
from decimal import Decimal
from typing import List, Set

@dataclass
class JobAnalysis:
    """Analysis of job postings for talent concentration."""
    total_ai_jobs: int
    senior_ai_jobs: int      # Principal, Staff, Director, VP level
    mid_ai_jobs: int         # Senior, Lead level
    entry_ai_jobs: int       # Junior, Associate, entry level
    unique_skills: Set[str]  # Distinct skills required

class TalentConcentrationCalculator:
    """
    Calculate talent concentration (key-person risk).

    Formula:
    TC = 0.4 * leadership_ratio + 0.3 * team_size_factor + 0.2 * skill_concentration + 0.1 *
    ↪ individual_mentions

    Where:
    - leadership_ratio = senior_jobs / total_jobs (0-1)
    - team_size_factor = 1 / sqrt(total_jobs) capped at 1 (smaller teams = higher TC)
    - skill_concentration = 1 - (unique_skills / 15) capped at [0,1]
    - individual_mentions = glassdoor mentions / reviews (0-1)

    Bounded to [0, 1]
```

```python
    """

    def calculate_tc(
        self,
        job_analysis: JobAnalysis,
        glassdoor_individual_mentions: int = 0,
        glassdoor_review_count: int = 1,
    ) → Decimal:
        """
        Calculate talent concentration ratio.

        Args:
            job_analysis: Analysis of job postings
            glassdoor_individual_mentions: Count of reviews mentioning specific people
            glassdoor_review_count: Total Glassdoor reviews

        Returns:
            Talent concentration ratio in [0, 1]
        """
        # TODO: Calculate leadership ratio
        # if job_analysis.total_ai_jobs > 0:
        #     leadership_ratio = job_analysis.senior_ai_jobs / job_analysis.total_ai_jobs
        # else:
        #     leadership_ratio = 0.5  # Default if no data

        # TODO: Calculate team size factor
        # team_size_factor = min(1.0, 1.0 / (job_analysis.total_ai_jobs ** 0.5 + 0.1))

        # TODO: Calculate skill concentration
        # skill_concentration = max(0, 1 - len(job_analysis.unique_skills) / 15)

        # TODO: Calculate individual mention factor
        # if glassdoor_review_count > 0:
        #     individual_factor = min(1.0, glassdoor_individual_mentions / glassdoor_review_count)
        # else:
        #     individual_factor = 0.5

        # TODO: Weighted combination
        # tc = (0.4 * leadership_ratio +
        #       0.3 * team_size_factor +
        #       0.2 * skill_concentration +
        #       0.1 * individual_factor)

        # TODO: Bound to [0, 1]
        # return Decimal(str(max(0, min(1, tc)))).quantize(Decimal("0.0001"))

        raise NotImplementedError("Implement calculate_tc")

    def analyze_job_postings(
        self,
        postings: List[dict],  # From CS2 job collector
    ) → JobAnalysis:
        """
        Categorize job postings by level.

        Senior keywords: principal, staff, director, vp, head, chief
        Mid keywords: senior, lead, manager
        Entry keywords: junior, associate, entry, intern
        """
        # TODO: Implement job categorization
        raise NotImplementedError("Implement analyze_job_postings")
```

7 Task 5.1: Decimal Utilities

scoring/utils.py — Provided + Stubs

```python
from decimal import Decimal, ROUND_HALF_UP
from typing import List

def to_decimal(value: float, places: int = 4) -> Decimal:
    """Convert float to Decimal with explicit precision."""
    return Decimal(str(value)).quantize(
        Decimal(10) ** -places, rounding=ROUND_HALF_UP)

def clamp(value: Decimal,
          min_val: Decimal = Decimal(0),
          max_val: Decimal = Decimal(100)) -> Decimal:
    """Clamp value to range [min_val, max_val]."""
    return max(min_val, min(max_val, value))

# TODO: Implement these
def weighted_mean(values: List[Decimal], weights: List[Decimal]) -> Decimal:
    """Calculate weighted mean."""
    raise NotImplementedError()

def weighted_std_dev(values: List[Decimal], weights: List[Decimal],
                     mean: Decimal) -> Decimal:
    """Calculate weighted standard deviation."""
    raise NotImplementedError()

def coefficient_of_variation(std_dev: Decimal, mean: Decimal) -> Decimal:
    """Calculate CV with zero-division protection."""
    raise NotImplementedError()
```

8 Task 5.2: V^{R} Calculator

See the V^{R} formula box on page 1. Key implementation points:

- Use sector-specific weights from `SectorConfigService`
- Apply non-compensatory CV penalty: penalty $= 1 - 0.25 \times \mathrm{cv}_D$
- Use **corrected** TalentRiskAdj: $1 - 0.15 \times \max(0, TC - 0.25)$
- Log all calculations with `structlog` for audit trail

9 Task 5.3: Property-Based Tests

Required Hypothesis tests for V^{R}:

1. `test_vr_always_bounded`: $0 \leq V^R \leq 100$ for all valid inputs
2. `test_higher_scores_increase_vr`: Monotonicity
3. `test_talent_concentration_penalty`: Higher TC $\Rightarrow$ lower V^{R}
4. `test_uniform_dimensions_no_cv_penalty`: Uniform scores $\Rightarrow$ penalty ≈ 1
5. `test_deterministic`: Same inputs $\Rightarrow$ identical output

NEW property tests for mapper:

1. `test_all_dimensions_returned`: Always returns 7 dimensions
2. `test_missing_evidence_defaults_to_50`: No evidence $\Rightarrow$ score $= 50$
3. `test_more_evidence_higher_confidence`: More sources $\Rightarrow$ higher confidence

Part II

Lab 6: H^R, Synergy & Full Pipeline

10 Lab 6 Preamble

Date	February 13, 2026 (Thursday)
Focus	Position factor, H^R, Synergy, SEM-based CI, Full integration
Time Estimate	Lecture: 2 hrs — Lab: 6 hrs — Challenge: +3 hrs

11 Task 6.0a: Position Factor Calculator [NEW]

NEW:

What is Position Factor? Position Factor measures a company's position relative to sector peers.

- PF = +1.0: Clear industry leader (e.g., NVDA in semiconductors)
- PF = 0.0: Average position
- PF = -1.0: Industry laggard

H^R uses position factor: $H^R = H^R_{base} \times (1 + 0.15 \times \text{PF})$

scoring/position_factor.py — YOUR IMPLEMENTATION

```python
from decimal import Decimal
from typing import Dict

class PositionFactorCalculator:
    """
    Calculate position factor for H^R.

    Formula:
    PF = 0.6 * VR_component + 0.4 * MCap_component

    Where:
    - VR_component = (vr_score - sector_avg_vr) / 50, clamped to [-1, 1]
    - MCap_component = (market_cap_percentile - 0.5) * 2

    Bounded to [-1, 1]
    """

    # Sector average V^R scores (from framework calibration data)
    SECTOR_AVG_VR: Dict[str, float] = {
        "technology": 65.0,
        "financial_services": 55.0,
        "healthcare": 52.0,
        "business_services": 50.0,
        "retail": 48.0,
        "manufacturing": 45.0,
    }

    def calculate_position_factor(
        self,
        vr_score: float,
        sector: str,
        market_cap_percentile: float,  # 0.0 = smallest, 1.0 = largest in sector
    ) → Decimal:
```

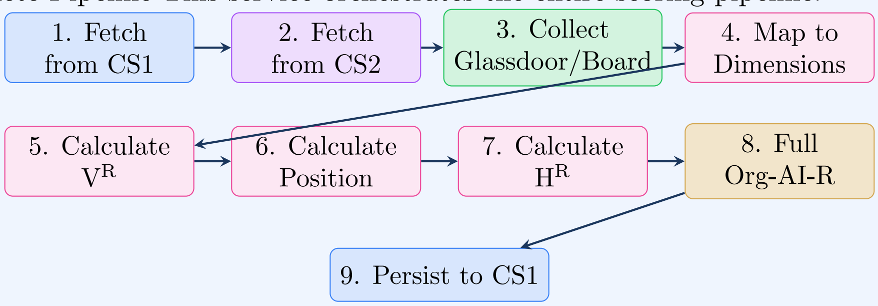


```python
        """
        Calculate position factor from V^R and market cap.

        Args:
            vr_score: Company's V^R score (0-100)
            sector: Company sector
            market_cap_percentile: Position in sector by market cap (0-1)

        Returns:
            Position factor in [-1, 1]
        """
        # TODO: Get sector average V^R
        # sector_avg = self.SECTOR_AVG_VR.get(sector.lower(), 50.0)

        # TODO: Calculate VR component
        # vr_diff = vr_score - sector_avg
        # vr_component = max(-1, min(1, vr_diff / 50))

        # TODO: Calculate market cap component
        # mcap_component = (market_cap_percentile - 0.5) * 2

        # TODO: Weighted combination
        # pf = 0.6 * vr_component + 0.4 * mcap_component

        # TODO: Bound to [-1, 1] and return as Decimal

        raise NotImplementedError("Implement calculate_position_factor")
```


12 Task 6.0b: Full Pipeline Integration Service [NEW]

CS1/CS2 Integration:

The Complete Pipeline This service orchestrates the entire scoring pipeline:



scoring/integration_service.py — YOUR IMPLEMENTATION

```python
import httpx
import structlog
from typing import Dict, Any
from decimal import Decimal

from scoring.evidence_mapper import EvidenceMapper, EvidenceScore, SignalSource
from scoring.rubric_scorer import RubricScorer
from scoring.talent_concentration import TalentConcentrationCalculator
from scoring.position_factor import PositionFactorCalculator
from scoring.vr_calculator import VRCalculator
from scoring.hr_calculator import HRCalculator
from scoring.synergy_calculator import SynergyCalculator
from scoring.confidence import ConfidenceCalculator
from pipelines.glassdoor_collector import GlassdoorCultureCollector
from pipelines.board_analyzer import BoardCompositionAnalyzer

logger = structlog.get_logger()

class ScoringIntegrationService:
    """
    Full pipeline from CS1/CS2 data to Org-AI-R score.

    This is the main entry point that orchestrates all components.
    """

    def __init__(
        self,
        cs1_api_url: str = "http://localhost:8000",
        cs2_api_url: str = "http://localhost:8001",
    ):
        self.cs1_url = cs1_api_url
        self.cs2_url = cs2_api_url

        # Initialize all components
        self.evidence_mapper = EvidenceMapper()
        self.rubric_scorer = RubricScorer()
        self.tc_calculator = TalentConcentrationCalculator()
        self.pf_calculator = PositionFactorCalculator()
        self.vr_calculator = VRCalculator()
        self.hr_calculator = HRCalculator()
        self.synergy_calculator = SynergyCalculator()
        self.ci_calculator = ConfidenceCalculator()
        self.glassdoor_collector = GlassdoorCultureCollector()
        self.board_analyzer = BoardCompositionAnalyzer()

        self.http = httpx.Client(timeout=30.0)
```

```python

    def score_company(self, ticker: str) → Dict[str, Any]:
        """
        Full scoring pipeline for a company.

        Returns complete assessment with audit trail.
        """
        logger.info("scoring_started", ticker=ticker)

        # Step 1: Fetch company from CS1
        company = self._fetch_company(ticker)
        logger.info("company_fetched", company_id=company["id"])

        # Step 2: Fetch evidence from CS2
        cs2_evidence = self._fetch_cs2_evidence(company["id"])
        logger.info("cs2_evidence_fetched", signal_count=len(cs2_evidence.get("signals", [])))

        # Step 3: Collect new CS3 data (Glassdoor, Board)
        glassdoor_signal = self._collect_glassdoor(company["id"], ticker)
        board_signal = self._collect_board(company["id"], ticker)

        # Step 4: Map all evidence to dimensions
        evidence_scores = self._build_evidence_scores(cs2_evidence, glassdoor_signal, board_signal)
        dimension_scores = self.evidence_mapper.map_evidence_to_dimensions(evidence_scores)

        # Step 5: Calculate talent concentration
        job_analysis = self.tc_calculator.analyze_job_postings(
            cs2_evidence.get("job_postings", [])
        )
        tc = self.tc_calculator.calculate_tc(
            job_analysis,
            glassdoor_individual_mentions=glassdoor_signal.get("individual_mentions", 0),
            glassdoor_review_count=glassdoor_signal.get("review_count", 1)
        )

        # Step 6: Calculate V^R
        vr_result = self.vr_calculator.calculate(
            dimension_scores={d.value: float(s.score) for d, s in dimension_scores.items()},
            talent_concentration=float(tc),
            sector=company["sector"]
        )

        # Step 7: Calculate position factor
        pf = self.pf_calculator.calculate_position_factor(
            vr_score=float(vr_result.vr_score),
            sector=company["sector"],
            market_cap_percentile=company.get("market_cap_percentile", 0.5)
        )

        # Step 8: Calculate H^R
        hr_result = self.hr_calculator.calculate(
            sector=company["sector"],
            position_factor=float(pf)
        )

        # Step 9: Calculate Synergy
        synergy_result = self.synergy_calculator.calculate(
            vr_score=vr_result.vr_score,
            hr_score=hr_result.hr_score,
            alignment=self._calculate_alignment(vr_result, hr_result),
            timing_factor=1.0  # Could adjust based on economic conditions
        )

        # Step 10: Calculate full Org-AI-R
        alpha = Decimal("0.60")
        beta = Decimal("0.12")

        weighted_components = alpha * vr_result.vr_score + (1 - alpha) * hr_result.hr_score
```

```python
        final_score = (1 - beta) * weighted_components + beta * synergy_result.synergy_score

        # Step 11: Calculate SEM-based confidence interval
        total_evidence = sum(es.evidence_count for es in evidence_scores)
        ci = self.ci_calculator.calculate(
            score=final_score,
            score_type="org_air",
            evidence_count=total_evidence
        )

        # Step 12: Build result
        result = {
            "company_id": company["id"],
            "ticker": ticker,
            "sector": company["sector"],
            "final_score": float(final_score),
            "vr_score": float(vr_result.vr_score),
            "hr_score": float(hr_result.hr_score),
            "synergy_score": float(synergy_result.synergy_score),
            "ci_lower": float(ci.ci_lower),
            "ci_upper": float(ci.ci_upper),
            "talent_concentration": float(tc),
            "position_factor": float(pf),
            "dimension_scores": {d.value: float(s.score) for d, s in dimension_scores.items()},
            "evidence_count": total_evidence,
            "confidence": float(ci.confidence),
        }

        # Step 13: Persist to CS1
        self._persist_assessment(result)

        logger.info("scoring_completed", ticker=ticker, final_score=result["final_score"])
        return result

    # TODO: Implement helper methods
    def _fetch_company(self, ticker: str) -> Dict:
        """Fetch company from CS1 API."""
        raise NotImplementedError()

    def _fetch_cs2_evidence(self, company_id: str) -> Dict:
        """Fetch evidence from CS2 API."""
        raise NotImplementedError()

    def _collect_glassdoor(self, company_id: str, ticker: str) -> Dict:
        """Collect Glassdoor data."""
        raise NotImplementedError()

    def _collect_board(self, company_id: str, ticker: str) -> Dict:
        """Collect board composition data."""
        raise NotImplementedError()

    def _build_evidence_scores(self, cs2_evidence: Dict,
                               glassdoor: Dict, board: Dict) -> list:
        """Build EvidenceScore list from all sources."""
        raise NotImplementedError()

    def _calculate_alignment(self, vr_result, hr_result) -> float:
        """Calculate alignment factor for synergy."""
        raise NotImplementedError()

    def _persist_assessment(self, result: Dict) -> None:
        """Persist assessment to CS1 API."""
        raise NotImplementedError()
```

13 Tasks 6.1-6.4: $\mathbf{H^R}$, SEM, Synergy, Org-AI-R

These tasks remain as in the previous version:

- **Task 6.1:** HRCalculator with corrected $\delta = 0.15$
- **Task 6.2:** ConfidenceCalculator with SEM (Spearman-Brown)
- **Task 6.3:** SynergyCalculator with TimingFactor $\in [0.8, 1.2]$
- **Task 6.4:** OrgAIRCalculator integrating all components

Key formulas:

$$H^R = H^R_{base} \times (1 + 0.15 \times \text{PositionFactor}) \quad (1)$$

$$\text{SEM} = \sigma \times \sqrt{1 - \rho}, \quad \rho = \frac{n \times r}{1 + (n-1) \times r} \quad (2)$$

$$\text{Synergy} = \frac{V^R \times H^R}{100} \times \text{Alignment} \times \text{TimingFactor} \quad (3)$$

14 Task 6.5: 5-Company Portfolio

Score these 5 real companies and validate against expected ranges:

Table 5: Target Company Portfolio with Expected Scores

Company	Sector	Expected	PF	TC	Why
NVIDIA	Technology	85-95	+0.9	0.12	AI chip leader
JPMorgan	Financial Svc	65-75	+0.5	0.18	$15B+ tech spend
Walmart	Retail	55-65	+0.3	0.20	Supply chain AI
General Electric	Manufacturing	45-55	0.0	0.25	Industrial IoT
Dollar General	Retail	35-45	-0.3	0.30	Limited tech

15 Deliverables Checklist

Deliverables Checklist

Lab 5 Deliverables (50 points):

- ☐ [NEW] Evidence Mapper with complete mapping table (10 pts)
- ☐ [NEW] Rubric Scorer with all 7 dimension rubrics (8 pts)
- ☐ [NEW] Glassdoor Culture Collector (7 pts)
- ☐ [NEW] Board Composition Analyzer (7 pts)
- ☐ [NEW] Talent Concentration Calculator (5 pts)
- ☐ Decimal utilities (3 pts)
- ☐ VRCalculator with audit logging (5 pts)
- ☐ Property-based tests (5 pts)

Lab 6 Deliverables (50 points):

- ☐ [NEW] Position Factor Calculator (5 pts)
- ☐ [NEW] Integration Service (full pipeline) (15 pts)
- ☐ HRCalculator with $\delta = 0.15$ (5 pts)
- ☐ SEM-based Confidence Calculator (5 pts)
- ☐ SynergyCalculator (5 pts)
- ☐ OrgAIRCalculator (5 pts)
- ☐ 5-company portfolio results (10 pts)

Testing Requirements:

- ☐ $\geq$ 80% code coverage
- ☐ All property tests pass with 500 examples
- ☐ Portfolio scores within expected ranges

16 Submission

Due: February 20, 2026 at 3:59 PM

```
cs3_scoring_engine/
   |-- src/scoring/
      |-- evidence_mapper.py          # NEW: Task 5.0a
      |-- rubric_scorer.py            # NEW: Task 5.0b
      |-- talent_concentration.py     # NEW: Task 5.0e
      |-- position_factor.py          # NEW: Task 6.0a
      |-- integration_service.py      # NEW: Task 6.0b
      |-- utils.py, vr_calculator.py, hr_calculator.py, ...
   |-- src/pipelines/
      |-- glassdoor_collector.py      # NEW: Task 5.0c
      +-- board_analyzer.py           # NEW: Task 5.0d
   |-- tests/
      |-- test_evidence_mapper.py, test_rubric_scorer.py, ...
   |-- results/
      |-- nvda.json, jpm.json, wmt.json, ge.json, dg.json
   +-- README.md
```

Designed by Sri Krishnamurthy, CFA · **sri@quantuniversity.com**