CREPID System Manual: Simple and Clear Explanation of How It Works

This manual explains the CREPID system, a tool that helps companies understand how their employees are performing, how their workload is distributed, whether they need training, if new hires are needed, who might need extra support, and who deserves raises or promotions. It uses simple data files about employees, their tasks, and available training programs, along with some settings, to make these decisions. The goal is to make everything clear for someone who doesn't know technical terms, focusing on what happens, why, and where numbers come from. We'll use the example data you provided (about employees Alice, Bob, and Charlie) to show how it works.

---

1. What is CREPID?

CREPID is like a smart assistant for managers. It looks at:

- **Employee details** (like their ID, name, role, salary, and when they joined).
- **Tasks they do** (like coding, meetings, or documentation, and how often/important they are).
- **Training options** (like courses available to improve skills).
- **Settings** (rules or limits set by the company, like how much work is too much).

It produces a report (in JSON format) with six parts:

1. **Activities with Metrics**: Details about each employee's tasks, including calculations like how much value they create.
2. **Rebalance**: Suggestions to shift tasks between employees to balance workload.
3. **Training**: Suggestions for training programs to improve performance.
4. **Hiring**: Whether the company needs to hire new people and what they should do.
5. **Risks**: Which employees might need extra help or could be at risk of being let go.
6. **Appraisal**: Who should get a raise, promotion, or be flagged for review.

Let's go through each part of the system, explaining what it does, why, and how it uses the data.

---

2. Loading and Checking Data (load_data)

What It Does

This part takes the company's data files (like spreadsheets) and checks them to make sure everything makes sense before doing any calculations. It creates a "Model" that holds all the data neatly together.

What Data It Uses

- **Roster file**: Lists employees (e.g., EmpID, Name, Role, SalaryINR, DateOfJoining).
    - Example: Alice (EmpID 1) earns ₹800,000 and joined on a specific date.
- **Activities file**: Lists tasks each employee does, how often (TimeFreq, 0-7), how important (Importance, 0-7), and how well they do it (Points, 0-2).
    - Example: Alice does "Code Review" 5 times a week (TimeFreq=5), it's very important (Importance=6), but she's not great at it (Points=0.8).
- **Skills file**: Lists training programs (e.g., a "Code Review Program" costs ₹3,000 and can improve performance by 50%).
- **Settings**: Rules like "no one should have more than 140 units of workload" or "training is worth it if it gives at least 1x return on investment."

What Happens and Why

1. **Load the Files**: It reads the data files into tables. Think of this like opening spreadsheets in Excel.
2. **Check for Problems**:
    - **Are all employees in the activities file listed in the roster?** If Alice's tasks mention EmpID 1, but EmpID 1 isn't in the roster, it stops and says, "Hey, there's an unknown employee ID!" This prevents mistakes from using fake or wrong employee data.
    - **Are numbers in the right range?**
        - TimeFreq (0-7): Ensures tasks aren't listed as happening 10 times a week (impossible).
        - Importance (0-7): Keeps task priority reasonable (e.g., 6 means very important).
        - Points (0-2): Ensures performance scores make sense (0=bad, 1=okay, 2=excellent).
        - If any number is outside these ranges (e.g., TimeFreq=8), it stops and says, "Fix this number!"
    - **Does each employee have 7-10 tasks?** This ensures everyone has a reasonable number of tasks. If Alice has only 3 tasks, or 15, it stops and says, "Alice's task count is wrong!" Why? Too few tasks might mean missing data; too many might mean someone's overworked.

    - **Are salaries positive?** If Bob’s salary is ₹0 or negative, it stops and says, “Salaries must be positive!” This avoids errors in calculations that use salary.
3. **Output**: If all checks pass, it bundles the data into a “Model” (like a big folder) with roster, activities, skills, and settings, ready for other parts to use.

Why This Matters: These checks ensure the data is correct before making decisions. Wrong data (like a negative salary or missing employee) could lead to bad suggestions, like firing someone by mistake or planning training for a non-existent person.

---

3. Calculating Metrics (compute_metrics)

What It Does

This part takes the employee tasks and adds new numbers to them, like how much “impact” a task has, how much money it’s worth, and how well the employee is doing overall. These numbers help decide who’s doing well or needs help.

What It Uses

- The “Model” from load_data, especially the activities table (with EmpID, Activity, TimeFreq, Importance, Points) and roster (for SalaryINR).

What Happens and Why

For each task (e.g., Alice’s “Code Review”), it calculates:

1. **TIm (Task Impact)**: Multiply TimeFreq by Importance.
    - Example: Alice’s Code Review has TimeFreq=5 and Importance=6, so TIm = 5 × 6 = 30.
    - Why? This shows how much effort and priority a task has. A high TIm means it’s a big deal (takes a lot of time and is important).
2. **RelWeight (Relative Weight)**: For each employee, take their TIm for a task and divide it by their total TIm (sum of TIm for all their tasks).
    - Example: Alice’s total TIm (across all tasks) is 114. For Code Review, TIm=30, so RelWeight = 30 ÷ 114 ≈ 0.2632.
    - Why? This shows what fraction of the employee’s effort goes to that task. It helps compare tasks fairly across employees.
3. **DollarValue**: Multiply RelWeight by the employee’s SalaryINR.
    - Example: Alice’s SalaryINR=₹800,000. For Code Review, RelWeight=0.2632, so DollarValue = 0.2632 × ₹800,000 ≈ ₹210,526.
    - Why? This shows how much of the employee’s salary is “spent” on that task, based on their effort.

4. **NetValue**: Multiply DollarValue by Points.
    - Example: For Code Review, Alice's Points=0.8, so NetValue = ₹210,526 × 0.8 ≈ ₹168,421.
    - Why? This adjusts the task's value based on how well the employee does it. Low Points (like 0.8) mean they're not performing well, so the task's value is reduced.
5. **EmpTotalTI**: Sum of TIm for all tasks per employee.
    - Example: Alice's TIm values are 30 (Code Review), 15 (Unit Testing), etc., totaling 114.
    - Why? This shows the employee's total workload. Higher means they're busier.
6. **WPI (Weighted Performance Index)**: For each employee, sum their NetValue across all tasks, divide by the sum of DollarValue.
    - Example: For Alice, total NetValue ≈ ₹579,507, total DollarValue ≈ ₹721,491. WPI = ₹579,507 ÷ ₹721,491 ≈ 0.8035.
    - Why? WPI shows how well an employee performs for the money they're paid. WPI < 1 means they're underperforming; WPI > 1 means they're exceeding expectations.

Output

The activities table is updated with these new columns (TIm, RelWeight, DollarValue, NetValue, EmpTotalTI, WPI) and saved back into the Model. It prints "✅ Metrics computed successfully!" to confirm it worked.

Why This Matters: These numbers turn raw data (like how often someone does a task) into useful insights (like how much value they create). They help identify who's doing well (high WPI) or struggling (low WPI), and which tasks are eating up their time or money.

---

4. Rebalancing Workload (suggest_rebalance)

What It Does

This part suggests moving tasks between employees to make workloads fairer. For example, if Alice is overloaded and Bob is great at a task she struggles with, it might suggest giving some of her work to Bob.

What It Uses

- The Model's activities table (with TIm, Points, Importance).
- Settings like WorkloadMinTI (e.g., 100) and WorkloadMaxTI (e.g., 140).

What Happens and Why

1. **Group Tasks**: It looks at each task (e.g., “Code Review”) across all employees.
2. **Find Donors and Takers**:
    - **Donors**: Employees doing a task poorly (Points < 1.0) that’s important (Importance ≥ 4).
        - Example: Alice does Code Review with Points=0.8 and Importance=6, so she’s a donor.
        - Why? These employees could benefit from doing less of this task, as they’re not good at it.
    - **Takers**: Employees doing the same task well (Points ≥ 1.2).
        - Example: Bob does Code Review with Points=1.5, so he’s a taker.
        - Why? These employees can handle more of this task since they’re good at it.
3. **Suggest Transfers**:
    - For each donor-taker pair, it calculates how much workload (TIm) can be moved.
    - Rules:
        - Don’t move more than 10 TIm at once (to keep changes reasonable).
        - Don’t leave the donor with less than 5 TIm for that task (they still need to do some of it).
        - Don’t overload the taker beyond WorkloadMaxTI (e.g., 140).
    - Example: Alice (donor) has TIm=30 for Code Review. Bob (taker) has total TIm=135. It suggests moving 10 TIm to Bob, because:
        - 10 is less than 30-5=25 (Alice keeps at least 5).
        - Bob’s new total (135+10=145) is slightly over 140, but the code allows it in this step.
    - Output: A list like { "fromEmp": 1, "toEmp": 2, "Activity": "Code Review", "DeltaTIm": 10 }.
4. **Update Totals**: After each suggestion, it updates the employees’ total TIm to avoid over-assigning in the next step.

Output

A list of suggestions, like moving 10 units of Code Review from Alice to Bob. In the example JSON, it also suggests moving 5 units from Charlie to Bob for Code Review.

Why This Matters: This balances work so strong performers take on more high-value tasks, while reducing load on struggling employees, making the team more efficient.

---

5. Suggesting Training (suggest_training)

What It Does

This part suggests training programs for employees who aren't performing well on certain tasks, calculating whether the training is worth the cost.

What It Uses

- The Model's activities table (with DollarValue, Points) and skills table.
- Settings like TrainingROIMin (e.g., 1.0, meaning training should at least double its cost in value).

What Happens and Why

1. **Find Weak Tasks**: For each employee, look at tasks where Points < 2 (not perfect performance).
   - Example: Alice has Points=0.8 for Code Review, so it's a weak task.
2. **Match with Training**: Check if the task (e.g., "Code Review") matches a training program in the skills file (e.g., "CR Program").
   - Why? Training should target specific weaknesses.
3. **Calculate Costs and Benefits**:
   - **TrainingCost**: Pick the cheaper option between external (CostExternalPerPersonINR) and in-house (CostInhousePerSessionINR).
     - Example: For Code Review, CR Program costs ₹3,000 (external) or ₹5,000 (inhouse), so use ₹3,000.
   - **ExpectedGain**: Multiply DollarValue by ExpectedLift (how much the training improves performance, e.g., 50% or 0.5).
     - Example: Alice's Code Review DollarValue=₹210,526. ExpectedLift=0.5, so ExpectedGain = ₹210,526 × 0.5 = ₹105,263.
   - **ROI (Return on Investment)**: Calculate (ExpectedGain - TrainingCost) ÷ TrainingCost.
     - Example: ROI = (₹105,263 - ₹3,000) ÷ ₹3,000 ≈ 34.09.
   - **Recommendation**: If ROI ≥ TrainingROIMin (e.g., 1.0), suggest "Train"; else, "Skip (Low ROI)".
     - Example: ROI=34.09 > 1.0, so recommend "Train" for Alice's Code Review.
4. **Output**: A table with columns like EmpID, Activity, Program, SkillArea, DeficitValue (DollarValue), TrainingCost, ExpectedGain, ROI, Recommendation.
   - Example: For Alice's Code Review: { "EmpID": 1, "Activity": "Code Review", "Program": "CR Program", "SkillArea": "Software", "DeficitValue": 210526.32, "TrainingCost": 3000, "ExpectedGain": 105263.16, "ROI": 34.09, "Recommendation": "Train" }.

Why This Matters: Training can improve weak areas, but only if it's worth the cost. This ensures the company spends money wisely to boost performance.

---

6. Hiring Decisions (hiring_decision)

What It Does

This part checks if the team is overworked and suggests hiring new people if needed, including what tasks they should do and how much to pay them.

What It Uses

- The Model's activities (with EmpTotalTI) and roster (with SalaryINR).
- Settings like WorkloadMaxTI (e.g., 140), HireTargetTI (e.g., 135), IdealTI (e.g., 140).

What Happens and Why

1. **Check for Overload**:
    - Count employees whose EmpTotalTI > WorkloadMaxTI (e.g., 140).
    - Calculate the team's average EmpTotalTI.
    - If 2 or more employees are overloaded, or the team average > WorkloadMaxTI, hiring is needed.
    - Example: Alice (TIm=114), Bob (135), Charlie (70). Average=106.3 < 140, and no one is over 140, so no hiring needed.
2. **If No Hiring Needed**: Return { "HireNeeded": false, "NewHires": 0, "JD_Activities": [], "HireTargetTI": 135, "BudgetINR": 0 }.
3. **If Hiring Needed**:
    - **Calculate Shortfall**: Sum how much each employee's TIm exceeds 140.
        - Example: If Bob had TIm=150, excess=10. Total excess across team determines how many hires are needed.
    - **Number of Hires**: Divide total excess by HireTargetTI (e.g., 135) and round up.
        - Example: Excess=20, HireTargetTI=135, so NewHires = ceil(20 ÷ 135) = 1.
    - **Job Description (JD) Activities**: Pick the top 5 tasks with high TIm (workload) and low Points (poor performance).
        - Example: If Code Review has high TIm and low Points, it's included in JD_Activities.
    - **Budget**: Use the median salary from the roster, adjusted by HireTargetTI ÷ IdealTI.
        - Example: Median SalaryINR=₹750,000, HireTargetTI=135, IdealTI=140. Budget = ₹750,000 × (135 ÷ 140) ≈ ₹723,214.

    - Output: { "HireNeeded": true, "NewHires": 1, "JD_Activities": ["Code Review", ...], "HireTargetTI": 135, "BudgetINR": 723214 }.

Why This Matters: If the team is overworked, hiring new people can spread the load. The system picks tasks that need better performance and sets a fair salary based on current employees.

---

7. Identifying Risks (risk_flags)

What It Does

This part flags employees who might need extra support (PIP, or Performance Improvement Plan) or could be let go (Separation) based on their performance and whether training can help.

What It Uses

- The Model's activities (with WPI, Importance, Points, DollarValue) and roster (with Name, Role, DateOfJoining).
- Settings like TrainingROIMin (e.g., 1.0).
- Training suggestions (from suggest_training, if available).

What Happens and Why

For each employee:

1. **Get Basic Info**:
    - WPI: From activities (e.g., Alice's WPI=0.8).
    - Salary and Role: From roster (e.g., Alice's SalaryINR=₹800,000, Role=Dev).
    - Tenure: Calculate years worked from DateOfJoining to today (September 27, 2025).
        - Example: Alice joined 4 years ago (YearsWorked=4).
2. **Find Critical Deficits**: Tasks with Importance ≥ 4 and Points < 1.0.
    - Example: Alice has 6 such tasks (e.g., Code Review, Importance=6, Points=0.8).
    - Calculate HighImpGapINR: Sum of DollarValue × (1 - Points) for these tasks.
        - Example: For Code Review, DollarValue=₹210,526, Points=0.8, so gap = ₹210,526 × (1-0.8) = ₹42,105. Total for Alice ≈ ₹157,193.
3. **Check Training ROI**: If training suggestions exist, see if any critical deficits have ROI < TrainingROIMin.
    - Example: Alice's Code Review has ROI=34.09 > 1.0, so training is viable.
4. **Decide Flags**:
    - **Separation (Yes/No)**: If any critical deficit has low ROI (training not worth it), flag "Yes" and remark "High-impact deficits, low ROI → should be fired."

        - Example: Alice’s ROIs are high (34.09, 15.84, etc.), so Separation_Flag=“No”.
    - **PIP (Yes/No)**: If there are critical deficits but ROI is high, flag “Yes” and remark “High-impact deficits → needs training.”
        - Example: Alice has 6 deficits, high ROI, so PIP_Flag=“Yes”.
    - **No Issues**: If no deficits, both flags are “No” and remark is “No issues.”
        - Example: Bob has no deficits (all Points ≥ 1.2), so no flags.
5. **Output**: A table with EmpID, Name, Role, WPI, HighImpDeficits, HighImpGapINR, PIP_Flag, Separation_Flag, Remark.
    - Example: For Alice: { "EmpID": 1, "Name": "Alice", "Role": "Dev", "WPI": 0.8, "HighImpDeficits": 6, "HighImpGapINR": 157192.98, "PIP_Flag": "Yes", "Separation_Flag": "No", "Remark": "High-impact deficits → needs training" }.

Why This Matters: This helps identify employees struggling with important tasks. If training can fix it, they get a PIP; if not, the company might consider letting them go to avoid wasting resources.

---

8. Suggesting Appraisals (suggest_appraisal)

What It Does

This part suggests whether employees should get a raise, promotion, or be flagged for review based on their performance (WPI) and how long they’ve worked.

What It Uses

- The Model’s activities (with WPI) and roster (with Name, Role, DateOfJoining).

What Happens and Why

For each employee:

1. **Get WPI and Tenure**:
    - WPI: From activities (e.g., Bob’s WPI=1.49).
    - YearsWorked: Calculate from DateOfJoining to today.
        - Example: Bob joined 3.5 years ago.
2. **Decide Suggestion**:
    - **WPI < 0.9**: “Risk Flag (Needs Review)”.
        - Example: Alice and Charlie (WPI=0.8) get this.
        - Why? Low WPI means poor performance, so they need closer attention.
    - **0.9 ≤ WPI ≤ 1.1**: “Normal Increment (~5%)”.
        - Why? Average performance deserves a standard raise.

        - **WPI > 1.2 and YearsWorked > 2**: “Promotion / Incentive”.
            - Example: Bob (WPI=1.49, YearsWorked=3.5) gets this.
            - Why? Great performance plus experience justifies a big reward.
        - **WPI > 1.1**: “Incentive”.
            - Why? Good performance deserves a bonus, but not a promotion if tenure is short.
        - **Else**: “No Change”.
3. **Output**: A table with EmpID, Name, Role, WPI, YearsWorked, AppraisalSuggestion.
    - Example: For Bob: { "EmpID": 2, "Name": "Bob", "Role": "Dev", "WPI": 1.49, "YearsWorked": 3.5, "AppraisalSuggestion": "Promotion / Incentive" }.

Why This Matters: This rewards good performers (like Bob) with raises or promotions and flags struggling employees (like Alice and Charlie) for review, ensuring fair treatment.

---

9. Example Walkthrough with JSON Data

Let’s see how the example data (Alice, Bob, Charlie) produces the JSON output:

- **Activities with Metrics**:
    - Alice’s Code Review: TIm=5×6=30, RelWeight=30÷114≈0.2632, DollarValue=0.2632×₹800,000≈₹210,526, NetValue=₹210,526×0.8≈₹168,421, WPI=0.8035 (underperforming).
    - Bob’s Code Review: Points=1.5, WPI=1.49 (excellent).
    - Charlie’s Documentation: Points=0.6, WPI=0.8 (underperforming).
- **Rebalance**: Alice and Charlie are donors for Code Review (Points=0.8, 0.9), Bob is a taker (Points=1.5). Suggest moving 10 TIm from Alice and 5 from Charlie to Bob.
- **Training**: Alice’s Code Review has high ROI (34.09), so recommend “Train”. Same for others.
- **Hiring**: No one’s TIm > 140, team average=106.3 < 140, so no hiring needed.
- **Risks**: Alice and Charlie have 6 deficits each (e.g., Code Review, Points < 1), but high ROI, so PIP_Flag=“Yes”. Bob has no deficits, so no flags.
- **Appraisal**: Bob gets “Promotion / Incentive” (WPI=1.49, YearsWorked=3.5). Alice and Charlie get “Risk Flag” (WPI=0.8).

---

10. Why It All Works Together

The system is like a puzzle:

- **Load_data** ensures the puzzle pieces (data) are correct.
- **Compute_metrics** measures each piece (task value, employee performance).

- **Suggest_rebalance** rearranges pieces for a better fit (fair workload).
- **Suggest_training** adds new pieces (skills) where needed.
- **Hiring_decision** checks if more pieces (employees) are required.
- **Risk_flags** spots broken pieces (underperformers).
- **Suggest_appraisal** decides who gets a prize (raise/promotion) or needs fixing.

Each part uses the same data (Model) and builds on the previous steps, ensuring decisions are consistent and based on clear numbers.