# Smart Factory Use Case Matching Engine – Documentation

## 1. Overview

The Smart Factory Matching Engine automatically recommends relevant Smart Factory Use Cases based on customer assessments. It evaluates problem statements, strategic priorities, process selections, and maturity levels, then identifies matching Use Cases from a structured knowledge base.

## 2. Project Files

|  |  |
| --- | --- |
| File | Description |
| assessment\_db.csv | Main database of assessments and match results. Each row represents a single customer assessment, including processes, problems, impacts, and resulting top Use Cases. |
| uc\_problems\_db.json | Master library of Smart Factory Use Cases, including metadata such as name, processes, maturity level, impacts, and problem texts. |
| matching\_engine.py | Python script implementing the matching engine, handling matching logic, weighting, peer relations, inspection, and comparison modes. |

## 3. Structure of assessment\_db.csv

This CSV file stores all assessments collected from customers. Columns:

|  |  |
| --- | --- |
| Column | Description |
| CREATED\_AT | Timestamp (YYYY-MM-DD HH:MM:SS). Together with EMAIL, forms the primary key. |
| EMAIL | Email address identifying the customer. |
| PROCESSES | Semicolon-separated list of process steps selected by the customer (e.g. 'Produktionsplanung; Qualitätsmanagement'). |
| PROBLEM\_TEXTS | Normalized list of problem statements describing operational challenges. |
| MATURITY\_LEVELS | Desired maturity levels selected by the customer (e.g. 'Diagnostizierend; Vorhersagend; Autonom'). |
| IMPACT\_PRIORITIES | Strategic impact focus of the customer (e.g. 'Kosten: 8; Qualität: 6; Liefertreue: 5'). |
| OLD\_MATCHES\_SCORED | Historical top-10 Use Cases with their old algorithm scores. |
| MATCHES\_SCORED | New top-10 Use Cases computed by the current matching engine with additive scoring model. |

## 4. Structure of uc\_problems\_db.json

This JSON file defines all Smart Factory Use Cases. Each Use Case entry contains metadata used for matching and scoring. Example structure:

{  
 "id": 6,  
 "name": {"de": "Bedarfsvorhersage", "en": "Demand Forecasting"},  
 "processes": [  
 {"label": {"de": "Bedarfsmanagement", "en": "Demand Management"}},  
 {"label": {"de": "Produktionsplanung", "en": "Production Planning"}}  
 ],  
 "maturity\_level": {"label": {"de": "Vorhersagend", "en": "Predictive"}},  
 "impact": [  
 {"label": {"de": "Kosten"}, "value": 8},  
 {"label": {"de": "Qualität"}, "value": 5},  
 {"label": {"de": "Liefertreue"}, "value": 7}  
 ],  
 "problems\_tackled": [  
 {"problem\_text": {"de": "Nachfrageschwankungen führen zu ineffizienter Planung", "en": "Demand fluctuations cause inefficient planning"}},  
 {"problem\_text": {"de": "Manuelle Anpassung der Bedarfsplanung", "en": "Manual adjustment of demand plans"}}  
 ]  
}

The JSON typically contains 30–50 Use Cases. Each entry contributes problem texts for semantic matching, impact values for strategic weighting, and maturity/process tags for structural relevance.

## 5. Matching Logic

1. 1. Normalize problem statements for both assessment and Use Cases.
2. 2. Perform soft matching via token overlap and sentence prefix similarity.
3. 3. Build peer relations among Use Cases in the same process based on ≥30% Jaccard overlap.
4. 4. Compute maturity weights (bottom-up: simpler UCs are preferred).
5. 5. Compute impact weights aligning UC impact dimensions with customer strategic priorities (0.1–1.0 scale).
6. 6. Filter Use Cases by selected processes if provided.
7. 7. Compute additive final score: 0.5×Base + 0.25×Impact + 0.15×Maturity + 0.10×Process.
8. 8. Rank and output top 10 Use Cases per assessment to MATCHES\_SCORED.

## 6. Script Usage

Run the matching engine with different modes:

• Batch recomputation: python matching\_engine.py — recomputes missing MATCHES\_SCORED or all (--recompute-all).

• Inspect mode: python matching\_engine.py --inspect '<timestamp>' '<email>' — runs detailed match analysis for one assessment.

• Comparison mode: python matching\_engine.py --compare '<timestamp>' '<email>' — compares old and new results with visualization.

## 7. Scoring Weights

Final scoring weights:

• Base (problem similarity): 50%

• Impact alignment: 25%

• Maturity alignment: 15%

• Process relevance: 10%