Fastener Recommendation Expert System - First Draft Report

Problem

Task

The system supports selecting appropriate fastening methods for joining materials in construction, manufacturing, and DIY applications. It recommends specific fasteners based on material properties, strength requirements, environmental conditions, and practical constraints.

Users

Target users include:

• DIY enthusiasts requiring guidance for home projects

• Engineering students learning about fastening technology

• Product designers selecting connection methods

• Procurement specialists evaluating fastener options

Domain Complexity

The problem involves multi-dimensional decision-making considering:

• Material compatibility across 8 material types

• Strength requirements on ordinal scales (6 strength levels, 4 resistance levels)

• Environmental factors (moisture, chemicals, temperature, vibration)

• Practical constraints (curing time, permanence, tooling requirements)

• Context-dependent suggestions based on fact combinations

Expert

Identity

> Jasper please fill this

Expertise Domain

The expert knowledge encompasses:

• Mechanical fastening (bolts, screws, cable ties)

• Adhesive bonding (PVA glue, epoxy, silicone sealant)

• Thermal joining (welding)

• Material science principles governing compatibility

• Environmental resistance requirements

• Load-bearing capacity specifications

Knowledge Models

Problem Solving Model

The system implements a diagnostic inference pattern:

1. Gather facts through structured questioning (10 questions)

2. Apply forward-chaining inference to derive conclusions
3. Filter candidate fasteners using multi-criteria matching
4. Generate context-specific suggestions based on fact patterns
5. Present ranked recommendations with explanatory information

Domain Model

Core entities and relationships:

Entities:

• Question: Represents user inquiry (choice or boolean type)
• Fastener: Connection method with properties and metadata
• MaterialProperties: Multi-dimensional property specification
• Rule: Conditional inference statement
• SuggestionRule: Context-triggered advisory statement

Enumerations:

• FastenerCategory: adhesive, mechanical, thermal
• Strength: none, very_low, low, moderate, high, very_high
• Resistance: poor, fair, good, excellent
• Permanence: removable, semi_permanent, permanent
• Rigidity: flexible, semi_flexible, rigid

Relationships:

• Each Fastener has one MaterialProperties instance
• MaterialProperties define compatibility with material lists
• Rules map fact patterns to category recommendations or exclusions
• SuggestionRules apply conditionally to fastener categories

Rule Model

The knowledge base contains 37 identified knowledge elements:

Rules (7):

• IF material_type=metal AND mechanical_strength IN [high, very_high] THEN recommend [mechanical, thermal] (priority 10)
• IF moisture_exposure IN [outdoor, submerged] AND flexibility=true THEN require water_resistance:excellent, rigidity:flexible (priority 9)
• IF permanence=removable THEN require permanence:removable (priority 8)
• IF curing_time=immediate THEN exclude [adhesive] (priority 7)
• IF material_type=wood AND mechanical_strength IN [low, moderate] THEN recommend [adhesive, mechanical] (priority 6)
• IF material_type IN [glass, ceramic] THEN recommend [adhesive] (priority 6)
• IF load_type IN [vibration, heavy_dynamic] THEN require vibration_resistance:good (priority 5)

Suggestion Rules (13):

Context-triggered recommendations for specific fastener-fact combinations (e.g., stainless steel for outdoor bolts, pilot holes for wood screws).

Fasteners (7):

Concrete instances with full property specifications including tools, surface preparation, curing times.

Questions (10):

Structured inquiries covering material type, strength, environmental exposure, permanence, timing, load type, and special requirements.

Inference Type

Forward-chaining rule-based inference:

1. User answers populate fact base incrementally
2. Rules evaluate conditions against accumulated facts
3. Rules fire when all conditions satisfied, adding conclusions
4. Rules execute in priority order (highest first)
5. Conclusions guide fastener filtering using multi-criteria matching
6. Ordinal comparisons enforce minimum thresholds
7. Category filtering applies recommendations and exclusions
8. Property matching verifies required characteristics

The system employs data-driven inference - facts trigger applicable rules to derive conclusions rather than working backward from hypotheses.

User Interface

CLI Implementation

Interactive command-line interface (`cli_test.py`) providing:

• Sequential question presentation with input validation

• Support for question skipping

• Real-time debug state persistence after each answer

• Formatted recommendation display with property details

• Contextual suggestions per fastener

• Session restart capability

Web Implementation (NOT YET IMPLEMENTED!)

Flask-based web application providing:

• Browser-accessible question interface

• Form-based answer submission

• Visual presentation of recommendations

• Persistent session management

Functionality Justification

Dual interface approach serves different use cases:

• CLI enables rapid testing, automation, and debugging workflows

• Web interface provides accessible user experience for end users

• Both implementations share core inference engine ensuring consistency

Tools Used

• Python 3.12: Core implementation language

• uv: Package management and virtual environment handling

• Flask: Web framework for HTTP interface

• PyYAML: Human-readable debug output serialization

• pytest: Automated testing framework

Implementation follows object-oriented design with dataclass models ensuring type safety. Enum types enforce valid property values. JSON-based knowledge base enables non-programmer knowledge editing.

Validation

Testing Strategy

Comprehensive test suite with 89 automated tests across 4 test modules:

test_models.py (20 tests):

• Dataclass serialization/deserialization correctness

• Question, Fastener, MaterialProperties, Rule model validation

• Property type enforcement

• JSON round-trip (loading and saving) integrity

test_knowledge_base.py (19 tests):

• Knowledge base loading and saving operations

• Data persistence across serialization

• Content integrity verification

• Expected element counts and identifiers

test_inference_engine.py (33 tests):

• Fact management operations

• Rule evaluation logic

• Forward-chaining inference execution

• Fastener matching algorithms

• Suggestion generation

• End-to-end recommendation scenarios

test_ordinal_scales.py (17 tests):

• Ordinal scale definitions

• Enum value validation

• Comparison operator correctness

• Moisture-to-resistance mapping

Validation Approach

• Unit tests verify individual component behavior

• Integration tests validate end-to-end recommendation scenarios

• Fixture-based tests ensure consistency against knowledge base

• Property-based tests verify ordinal comparison logic

• Debug state YAML enables manual inspection of inference traces

Performance Verification

Test execution: 89 tests collected and executed via `uv run pytest`

All tests passing confirms:

• Rule evaluation produces expected conclusions

• Fastener filtering correctly applies criteria
• Ordinal comparisons enforce minimum thresholds
• Category filtering respects recommendations and exclusions

Task Division

This draft represents individual development. Knowledge engineering, system architecture, inference engine implementation, testing framework, and documentation were completed as integrated effort.

Future collaborative work will require:
• Knowledge base expansion through expert interviews
• User interface refinement based on testing feedback
• Additional test coverage for edge cases
• Performance optimization for larger knowledge bases

Repository Information

GitHub Repository: https://github.com/JasperK04/KTP
Branch: dev
Entry Point: `src/main.py` (web, not implemented) or `src/cli_test.py` (CLI)
Dependencies: Managed via `src/pyproject.toml`, installed with `uv sync`

Running the System

```
cd src
uv sync
uv run python cli_test.py # CLI interface
uv run python main.py # Web interface (not implemented)
uv run pytest # Test suite
```

Project Structure

• `src/engine.py` (474 lines): Core inference engine and knowledge base
• `src/kb.json` (396 lines): Declarative knowledge base
• `src/cli_test.py` (227 lines): CLI testing interface
• `src/tests/`: Automated test suite (89 tests)
• `src/README.md`: Usage documentation
• `src/cli_test.md`: Testing workflow and debug format documentation

Knowledge Complexity

Total identified knowledge elements: 37
• 10 Questions defining the fact space
• 7 Fastener instances with complete specifications
• 7 Inference rules with priority ordering
• 13 Suggestion rules for contextual recommendations

Each fastener contains 11 material properties, requires_tools list, surface_prep list, curing_time specification, and notes. This yields approximately 100+ distinct knowledge elements when counting individual property specifications.

The system meets the minimum complexity requirement of 100 knowledge elements through:

• 37 top-level knowledge entities

• 7 fasteners × 11 properties = 77 property specifications

• Multi-valued properties (e.g., compatible_materials lists)

• Contextual suggestion conditions and triggers

Total knowledge base size: 396 lines of structured JSON containing comprehensive domain knowledge about fastening technology.