# Design Of Expert System

2025-04-24 18:54

# Design Of Expert System

## 📚 Detailed Notes

# 📘 Chapter 6: Practical Development of Expert Systems

---

## 🔹 6.1 Introduction

- This chapter shifts from theory to **real-world expert system development**.
- Uses **software engineering methodologies** to ensure systems are **cost-effective**, **timely**, and **maintainable**.
- Expert system design is embedded in **Knowledge Management (KM)**, which intersects with:
  - **Information Management (IM)**
  - **Information Processing (IP)**
  - **Information Systems (IS)**
  - **Information Technology (IT)**

KM is vital due to the need to manage large, diverse, and growing information assets efficiently.

---

## 🔹 6.2 Selecting the Appropriate Problem

### ✅ Key Questions Before Development

1. **Why build it?**
   - Needs a business case and management backing.
   - Avoid IP disputes; most workplace-generated ideas belong to the company.
2. **Who is the expert?**
   - Stick to **single expert per system** to avoid complexity.

- Multi-expert systems require **blackboard architecture**.

3. **What is the payoff?**
    - ROI can be monetary, efficiency, or performance gains.
    - No use = No payoff.
4. **What tools should be used?**
    - Choose tools with proven success.
    - Understand **strong vs weak coupling**:
        - **Strong**: Rule A triggers Rule B (clear logic path).
        - **Weak**: Rule A enables multiple possible paths.
    - Tools like **CLIPS** and **Protégé** are common choices.
5. **How much will it cost?**
    - Hidden costs include:
        - Expert unavailability
        - Training
        - Maintenance (can be 10× development cost)
    - Use **public domain** or widely available knowledge to reduce cost.

---

## 🔷 6.3 Stages in Development

### 📌 Three Major Management Areas:

1. **Activity Management**: Planning, milestones, scheduling.
2. **Product Configuration**: Version control, change tracking.
3. **Resource Management**: Acquire and manage expert availability and tools.

### 🧭 Project Development Strategy

- Align schedules with expert availability.
- Commercial world emphasizes **marketing pre-releases** over internal QA.
- Use automatic bug reporting and public betas to gather real-world feedback.

### 🚚 System Delivery

- Deployment considerations include:
    - Platform independence (e.g., Java's "write once, run anywhere")
    - Integration with other programs (CLIPS can be called from C++)
    - Efficient communication between systems

## 🔹 6.4 Errors in Development

| Error Type | Details |
|---|---|
| **Expert Errors** | Inaccurate or outdated knowledge; panels and review boards (like NASA's Flight Panels) can help |
| **Semantic Errors** | Misinterpretation of expert's statements (e.g., "All fires can be extinguished by water") |
| **Syntax Errors** | Programming/formalization mistakes |
| **Inference Engine Errors** | Software bugs in pattern matching or conflict resolution |
| **Inference Chain Errors** | Incorrect priorities, rule conflicts, or propagation of uncertain/incomplete info |
| **Limits of Ignorance** | System may continue making poor conclusions unless explicitly told to stop |

🧠 *Cognitive illusions*, perception bias, and **contextual misunderstandings** are common and must be accounted for during expert interviews.

## 🔹 6.5 Software Engineering & Expert Systems

- **Expert systems** must meet software quality standards—especially for **mission-critical applications**.
- Must apply rigorous testing, especially when **human life or property is at stake**.

### 🧮 Software Quality Metrics (e.g., Table 6.1)

- Correctness
- Completeness
- Usability
- Robustness
- Maintainability
- Cost-effectiveness

👉 *Testing must balance accuracy, cost, and scheduling.*

## 🔹 6.6 The Expert System Life Cycle

### 💡 Life Cycle = Continuous Process:

From **initial concept** → **delivery** → **maintenance** → **retirement**

### 📈 Maintenance

- More expensive than initial development
- Expert systems require **continuous updates** due to evolving knowledge
- Tools like **Protégé** and automated ontologies help, but systems grow in size and complexity

---

## 🔁 Life Cycle Models

### 1. Waterfall Model

- Linear, stage-by-stage
- Good for stability, but lacks flexibility
- Backtracking is costly

### 2. Code-and-Fix

- Undisciplined, mostly for **student projects or prototypes**

### 3. Do-It-Twice

- Build prototype first → determine requirements → build final system

### 4. Incremental Model

- Build in functional increments:
  - **Assistant** → **Colleague** → **Expert**
  - Easier to test, validate, refine

### 5. Spiral Model

- Incremental + iterative
- Each loop adds functionality
- New spiral begins with maintenance

## 🔹 6.7 Detailed Life Cycle Model (Linear Model)

Follows sequence from **Planning** → **Evaluation**, then repeats for future improvements.

### ✅ Core Stages and Tasks

| Stage | Key Tasks |
|---|---|
| **Planning** | Feasibility, budgeting, scheduling (Table 6.2) |
| **Knowledge Definition** | Identify sources, acquire/analyze/extract (Tables 6.3, 6.4) |
| **Knowledge Design** | Define structure, detailed system blueprint (Tables 6.5, 6.6) |
| **Code & Checkout** | Implement code, test readiness (Table 6.7) |
| **Knowledge Verification** | Formal testing & result analysis (Tables 6.8, 6.9) |
| **System Evaluation** | Review performance, make recommendations (Table 6.10) |

## ✅ Final Takeaways

- ✅ **Expert systems development** must combine **AI principles** with **software engineering discipline**.
- ✅ Carefully **select problems**, tools, and experts.
- ✅ Plan for **uncertainty**, **rule conflicts**, and **maintenance** from the beginning.
- ✅ Follow a structured **life cycle model** to ensure quality and adaptability.
- ✅ Expert systems are never truly finished—they **evolve with knowledge** and user needs.

## 🔑 Key Terms

| Term | Definition |
|---|---|
| **Expert System (ES)** | A computer system that emulates the decision-making ability of a human expert. |
| **Knowledge Management (KM)** | Discipline focused on capturing, distributing, and effectively using knowledge. |

| Term | Definition |
|---|---|
| **Inference Engine** | Component of an expert system that applies rules to known facts to deduce new facts. |
| **Rule Coupling** | The logical connection between rules: **strong** (linear inference) vs **weak** (multiple outcomes). |
| **Semantic Net** | A network of terms/concepts showing relationships, used in knowledge representation. |
| **Ontology** | A structured framework for organizing information in a domain, used in ES for consistency. |
| **Life Cycle Model** | A framework describing all stages of development from concept to retirement of a system. |
| **Waterfall Model** | Sequential development model; each stage must be completed before the next begins. |
| **Incremental Model** | Development in increments or modules; each module adds functionality. |
| **Spiral Model** | Iterative model combining waterfall and prototyping approaches with risk assessment. |
| **Code-and-Fix Model** | Unstructured approach where coding is followed by debugging and correction. |
| **Knowledge Acquisition** | Process of gathering knowledge from experts or sources to build the knowledge base. |
| **Verification & Validation (V&V)** | Processes to ensure the system is built right (verification) and is the right system (validation). |
| **Semantic Error** | Misinterpretation of meaning in expert knowledge (e.g., overgeneralization of facts). |
| **Syntax Error** | Mistakes in formatting or writing of rules that prevent execution. |
| **Inference Chain Error** | Logical flaw or misstep in chaining together rules and conclusions. |
| **Maintenance** | Post-deployment updates to improve or correct the system based on evolving knowledge. |
| **Feasibility Study** | Preliminary analysis to determine if building the expert system is worthwhile. |
| **CLIPS** | A rule-based programming language used for building expert systems. |
| **Protégé** | Ontology and knowledge-base editor from Stanford used for building complex systems. |

## 💡 Key Points

- **Problem Selection** is crucial. Avoid multi-expert modeling unless necessary.
- **Return on Investment (ROI)** must be clear: money, time, efficiency.
- **Tool Choice** must balance cost, usability, and long-term maintenance.
- **Semantic and Ontological structuring** are key to efficient rule-based systems.
- **Development Stages** must be planned using structured life cycle models (Waterfall, Incremental, Spiral).
- **Errors** occur at multiple levels:
  - Expert level (false knowledge)
  - Knowledge engineer (misinterpretation)
  - Tool bugs, rule interaction, syntax, semantics
- **Maintenance** is the most expensive phase and must be planned early.
- **Validation and verification** are vital at every step.
- **Use automated tools** for knowledge extraction and ontology construction (e.g., Protégé, CLIPS).
- **Expert Systems are iterative and evolve**—they are never really "done."

---

## ✨ Summary

This chapter explains the **practical methodology for building real-world expert systems**, not just prototypes. It underscores the importance of applying **software engineering principles** to ensure high quality, cost-effective systems. The development begins with **problem selection**, goes through planning, knowledge acquisition, design, coding, testing, verification, and ends in evaluation.

The chapter introduces **life cycle models** such as the **Waterfall, Incremental, Spiral**, and the **Linear** model tailored to expert systems. Emphasis is placed on dealing with errors, planning for uncertainty, and handling the critical phase of **maintenance and evolution**. Real-world deployment issues and interoperability with other software systems are also discussed, with **CLIPS and Protégé** highlighted as key tools.

---

## ❓ Review Questions

### Objective Questions

1. What is the main goal of Knowledge Acquisition in expert systems?
2. What distinguishes a strong rule coupling from a weak one?
3. Which tool is used for ontology editing in expert systems?
4. Which life cycle model is based on functional increments of the system?

## ◆ True/False

1. The Code-and-Fix model is ideal for mission-critical expert systems. *(False)*
2. Expert systems should always be built using multiple experts in a single system. *(False)*
3. Maintenance costs are typically less than development costs. *(False)*
4. CLIPS can be integrated with C++ applications. *(True)*

## ◆ Short Answer

1. What are the main differences between the Waterfall and Spiral models?
2. Why is the feasibility study considered the most important stage?
3. What role do semantic errors play in expert system failures?

## ◆ Discussion

- Explain the importance of verification and validation in each stage of expert system development.
- Describe the stages and tasks in the Linear Life Cycle Model for expert systems.