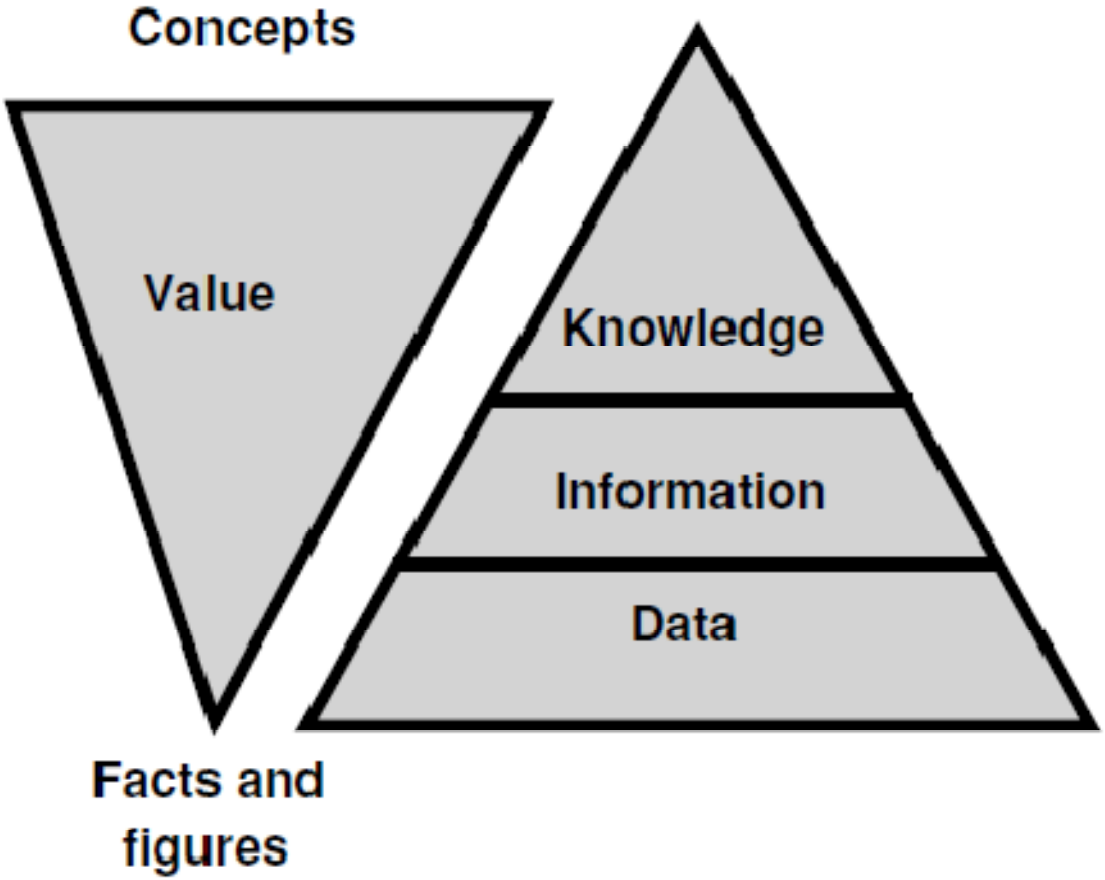
Part 1:
Knowledge and Expert Systems

Knowledge



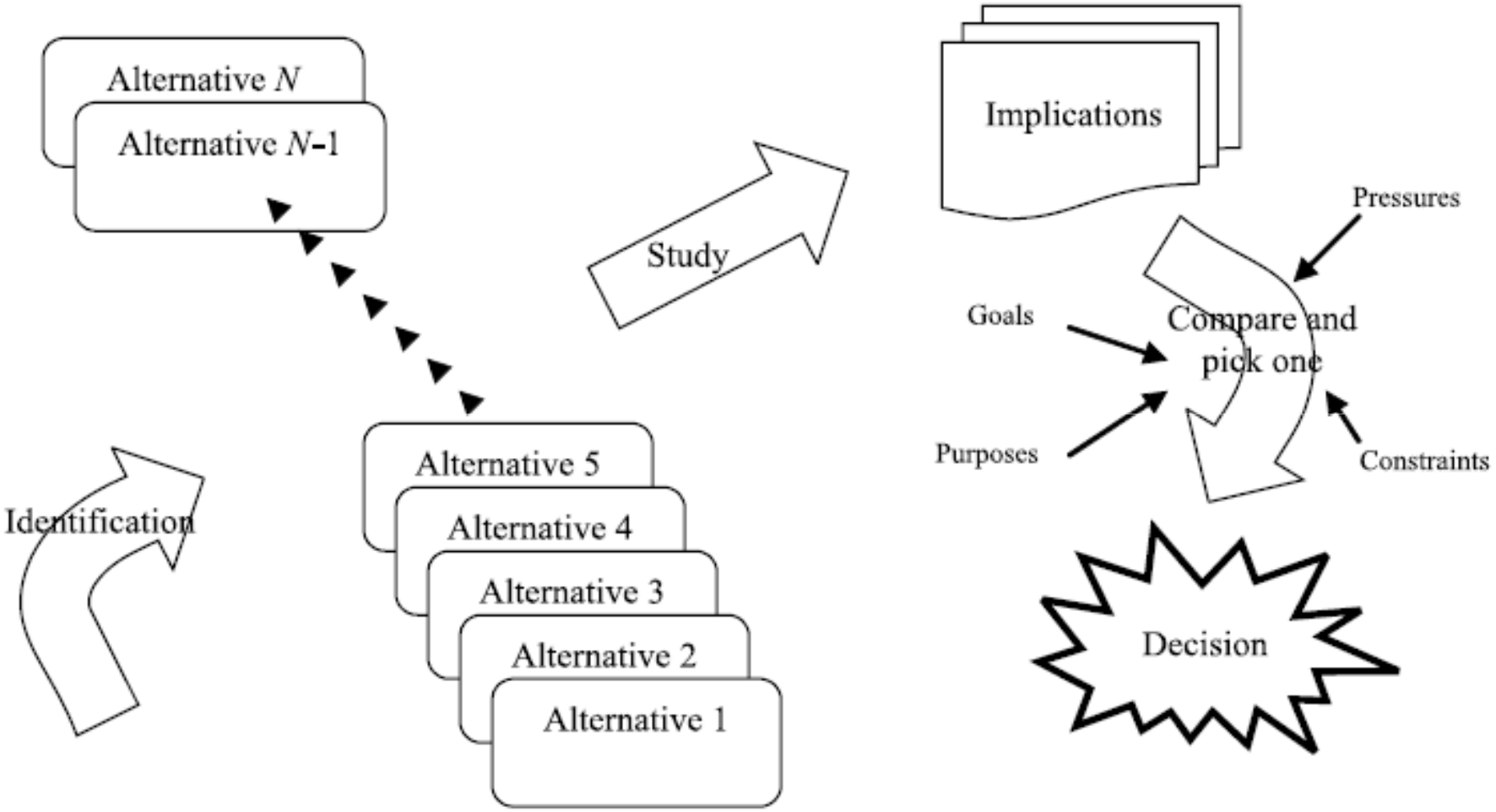
Example

It is cold – put on a warm coat.

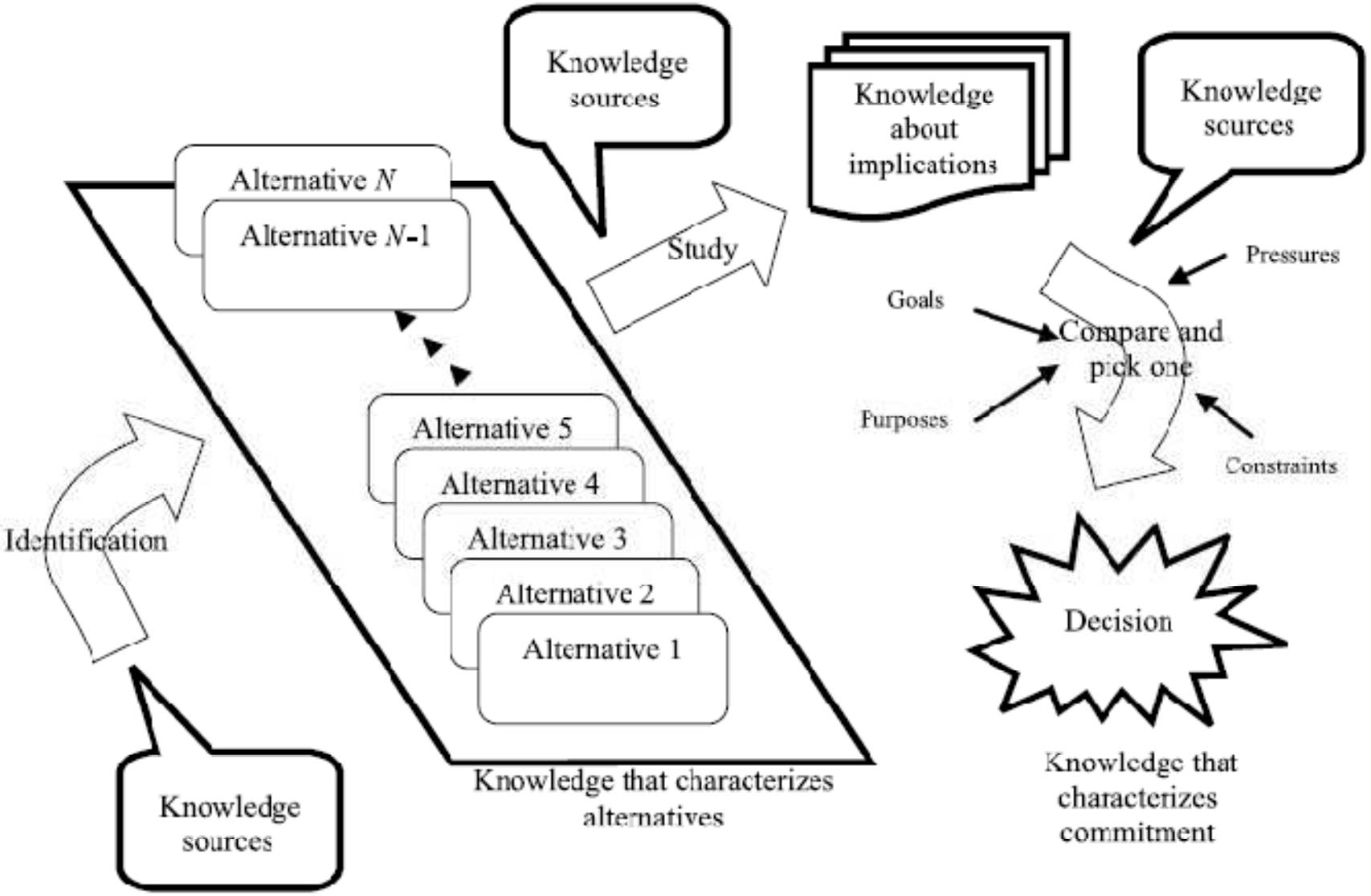
It is cold outside.

The temperature outside is 5°C

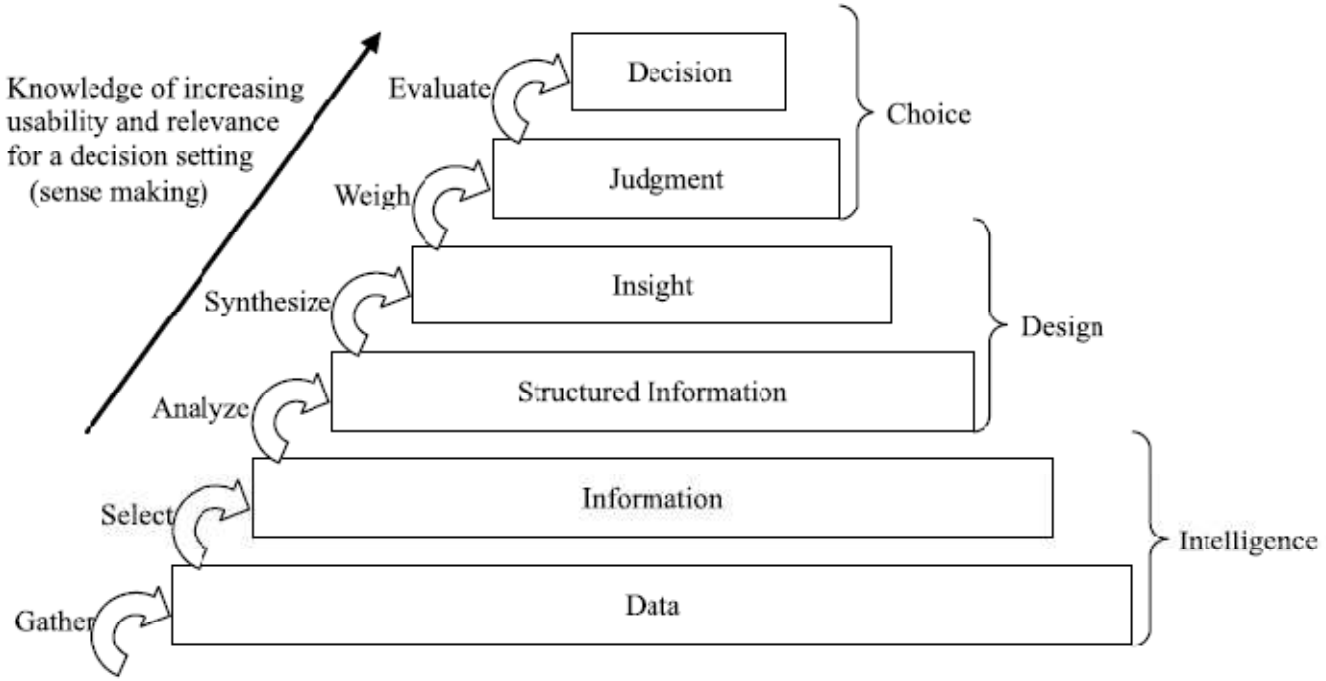
Traditional conception of decision making



Knowledge-based conception of decision making

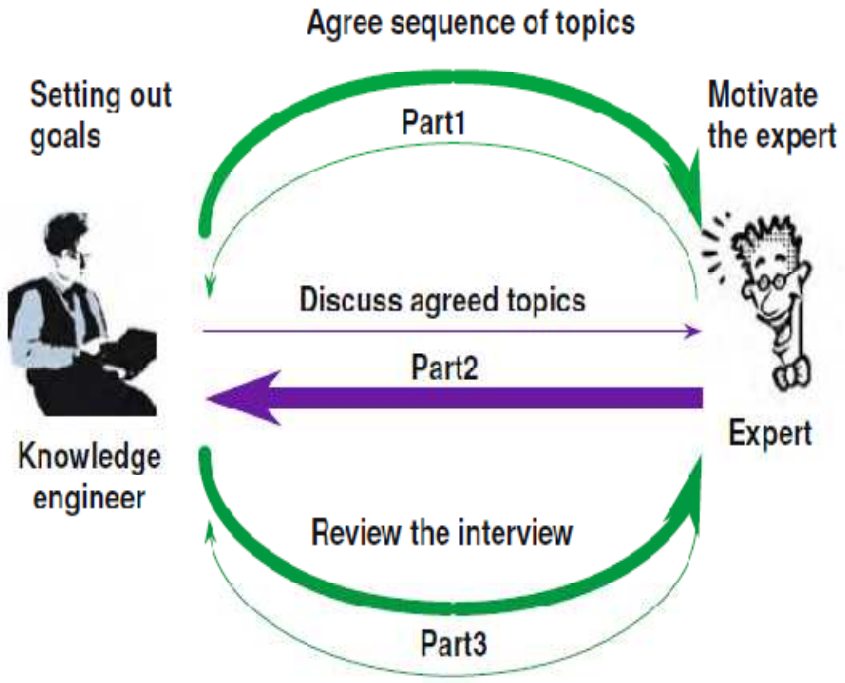


Knowledge as a progression of states



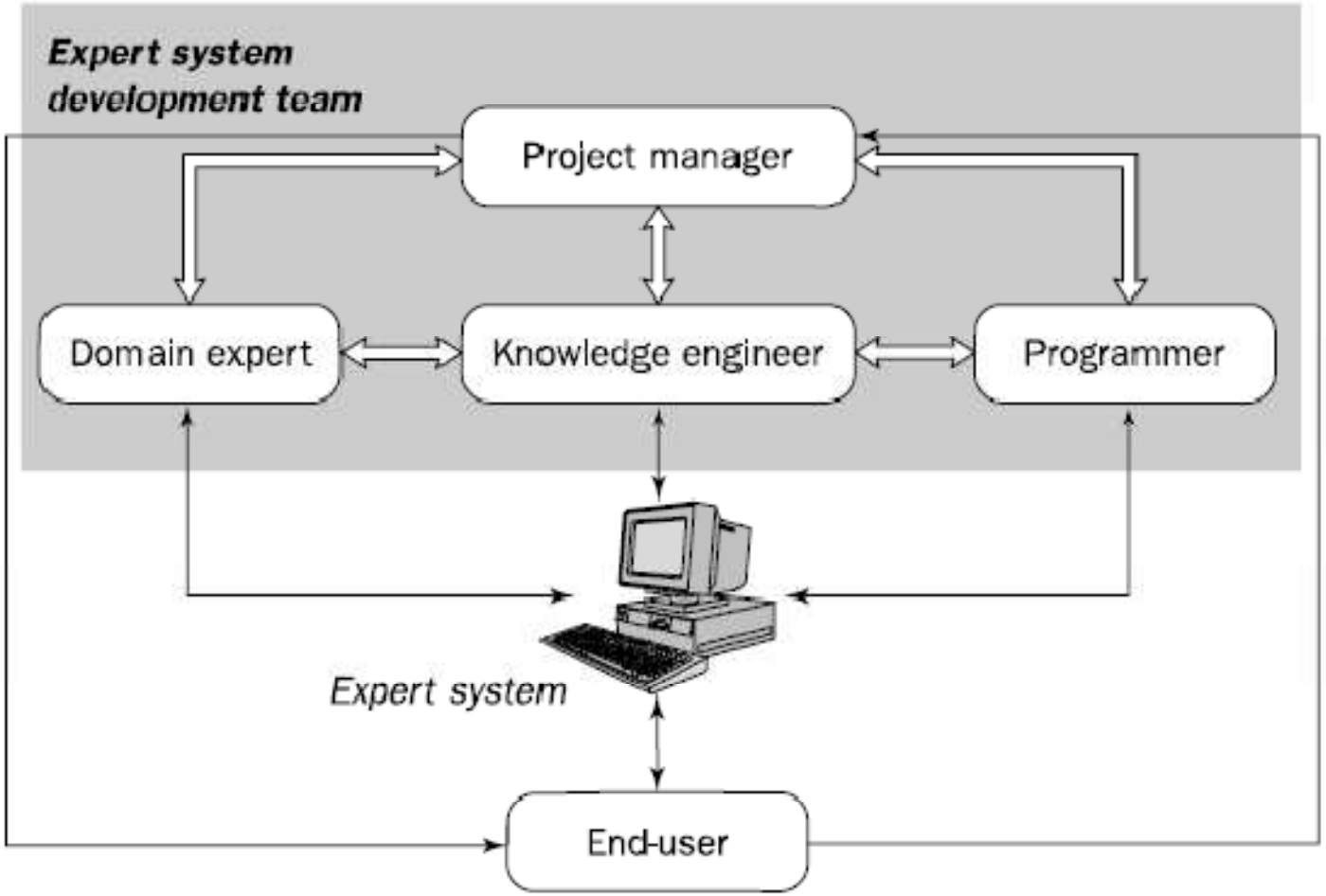
A progression of knowledge states	A sample progression
Datum	240
Information	240 is the level of cholesterol
Structured information	240 is the current level of cholesterol for John Doe
An evaluation	John Doe's level of cholesterol is now too high
A judgment	John Doe's health is presently in jeopardy
A decision	John Doe gets a prescription for Lipitor

Knowledge Acquisition

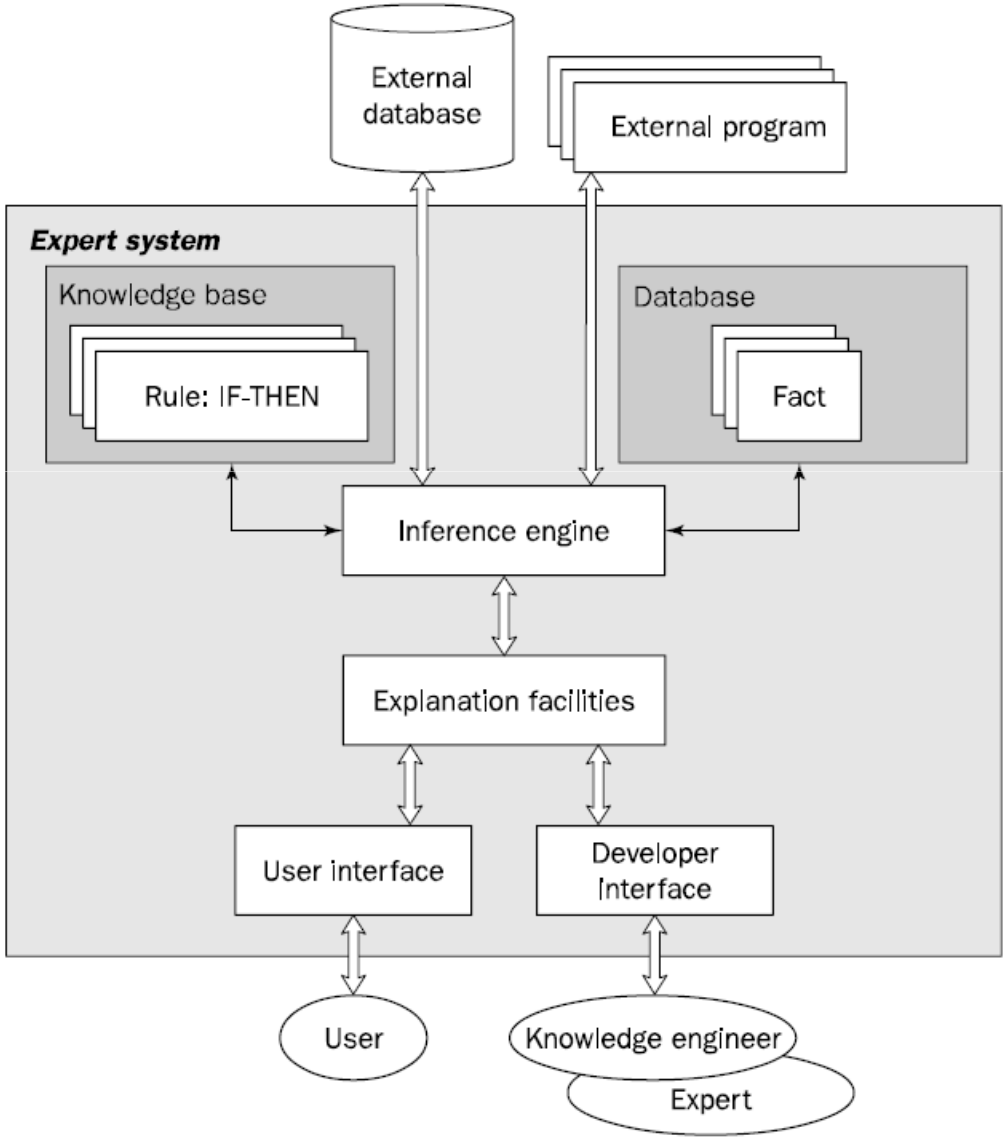


- Can you give me an overview of the subject?
- Can you describe the last case you dealt with?
- What facts or hypotheses do you try to establish when thinking about a problem?
- What kinds of things do you like to know about when you begin to think about a problem?
- Leading on to find a little more detail; tell me more about how this is achieved?
- What do you do next?
- How does that relate to . . . ?
- How, why, when, do you do that?
- Can you describe what you mean by that?

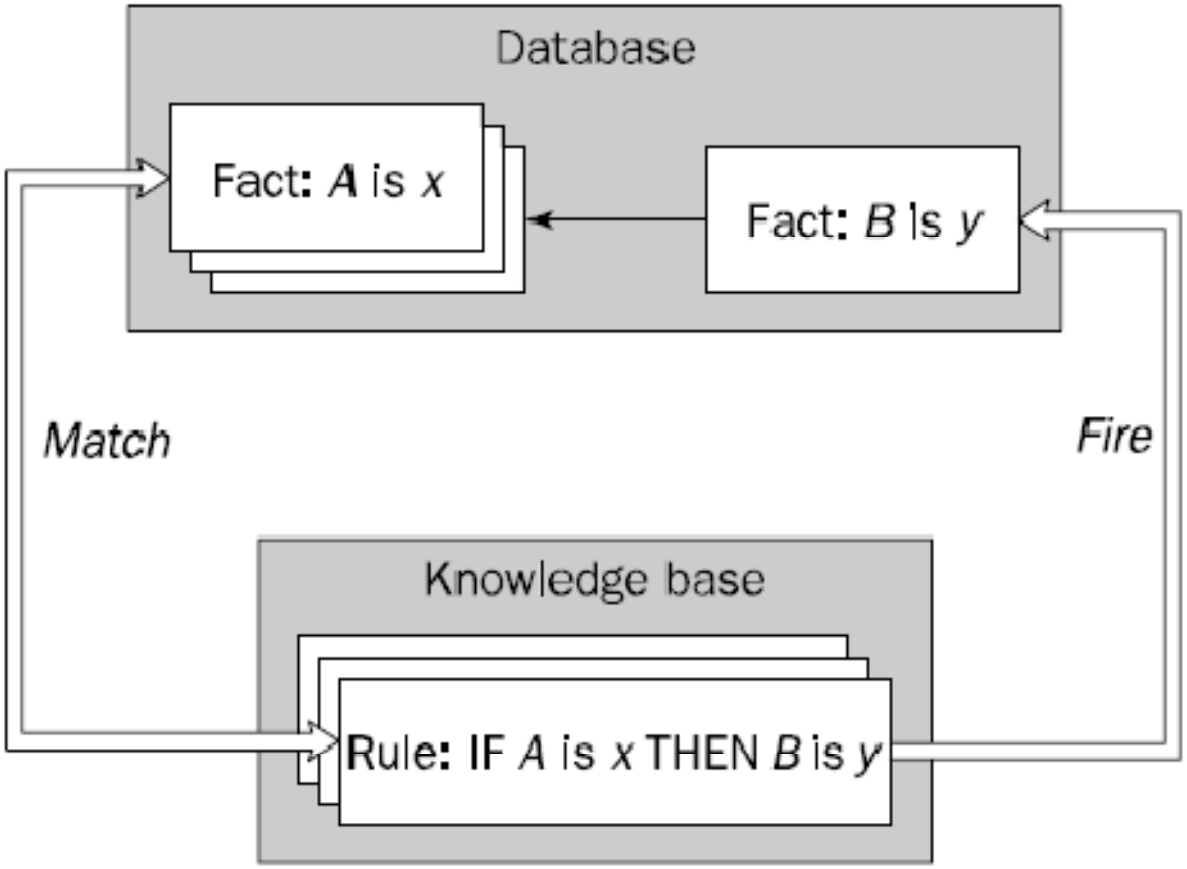
The role of the Knowledge Engineer



Complete structure of a rule-based expert system

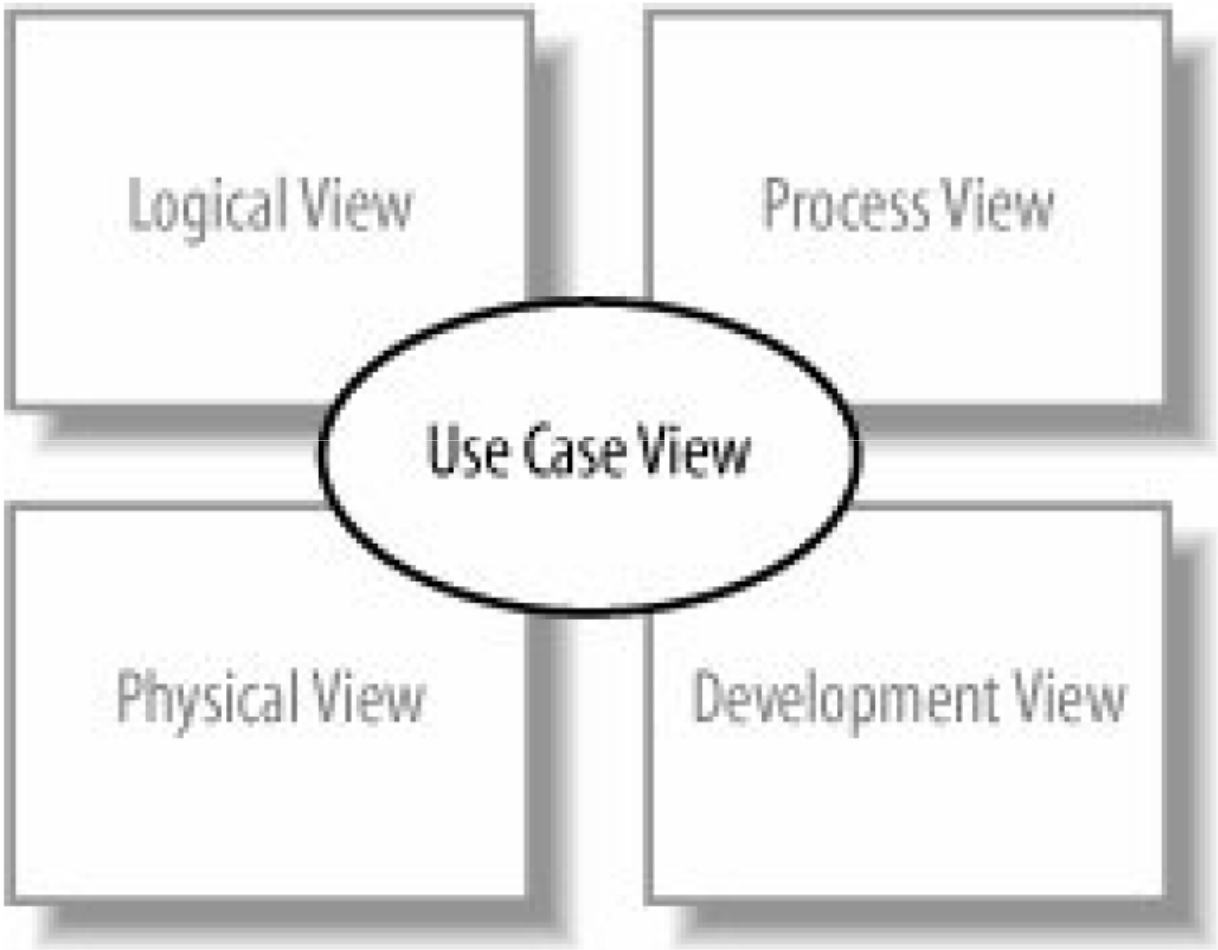


The inference engine cycles via a match-fire procedure



Rule 1:	IF	*Y* is true
	AND	*D* is true
	THEN	*Z* is true
Rule 2:	IF	*X* is true
	AND	*B* is true
	AND	*E* is true
	THEN	*Y* is true
Rule 3:	IF	*A* is true
	THEN	*X* is true

Part 2:
Conceptual Modeling of Information Systems


Logical View
Process View
Use Case View
Physical View
Development View

Use Cases

- Use cases are **logical models**
  - They describe the activities of a system
  - Do not address implementation details

- Steps to create a use case:
  - Gather requirements
  - Prepare Use Case Diagram(s)
  - Prepare Use Case Description for each Use Case

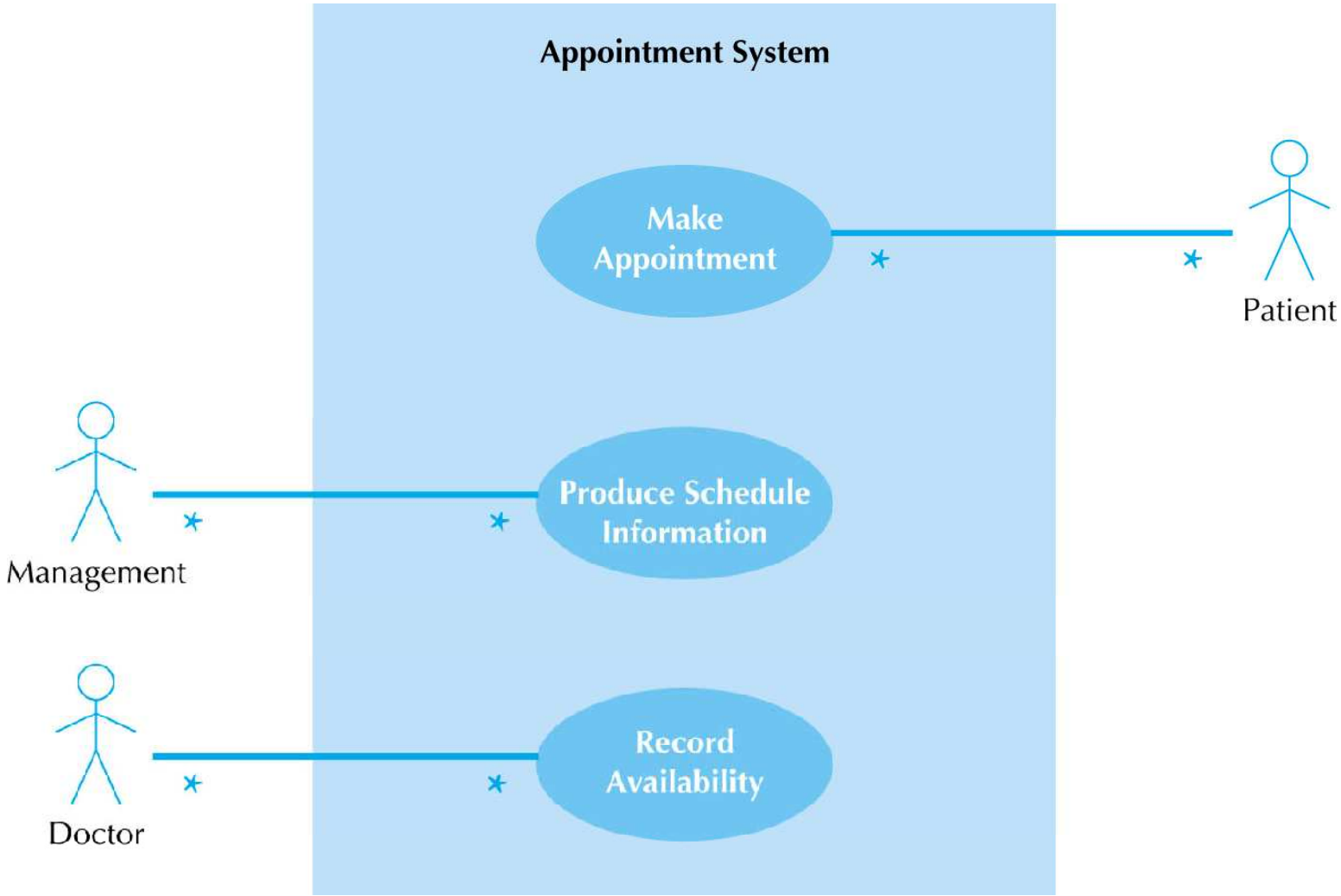
How Are Use-Cases Created?

- Each Use Case describes one and only one function
- But may have several paths that the user can take to accomplish that single function
- Developed working with users for content
  - Use Interviews, JAD, and observation

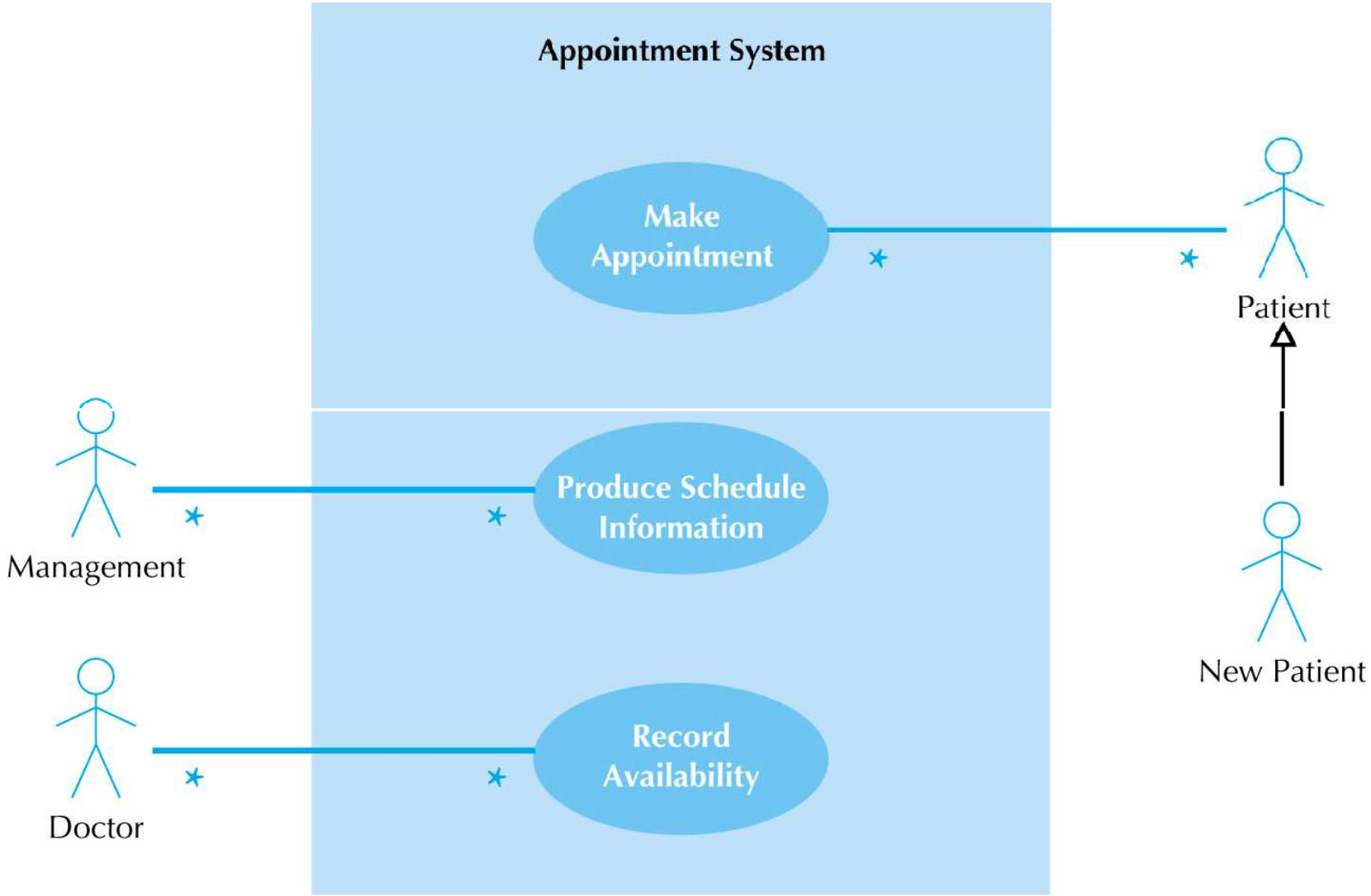
Use Case Diagram Syntax

- Actor
  - person or system that derives benefit from and is external to the subject
- Use Case
  - Represents a major piece of system functionality
- Association Relationship
- Include Relationship
- Extend Relationship
- Generalization Relationship

Use-Case Diagram for Appointment System



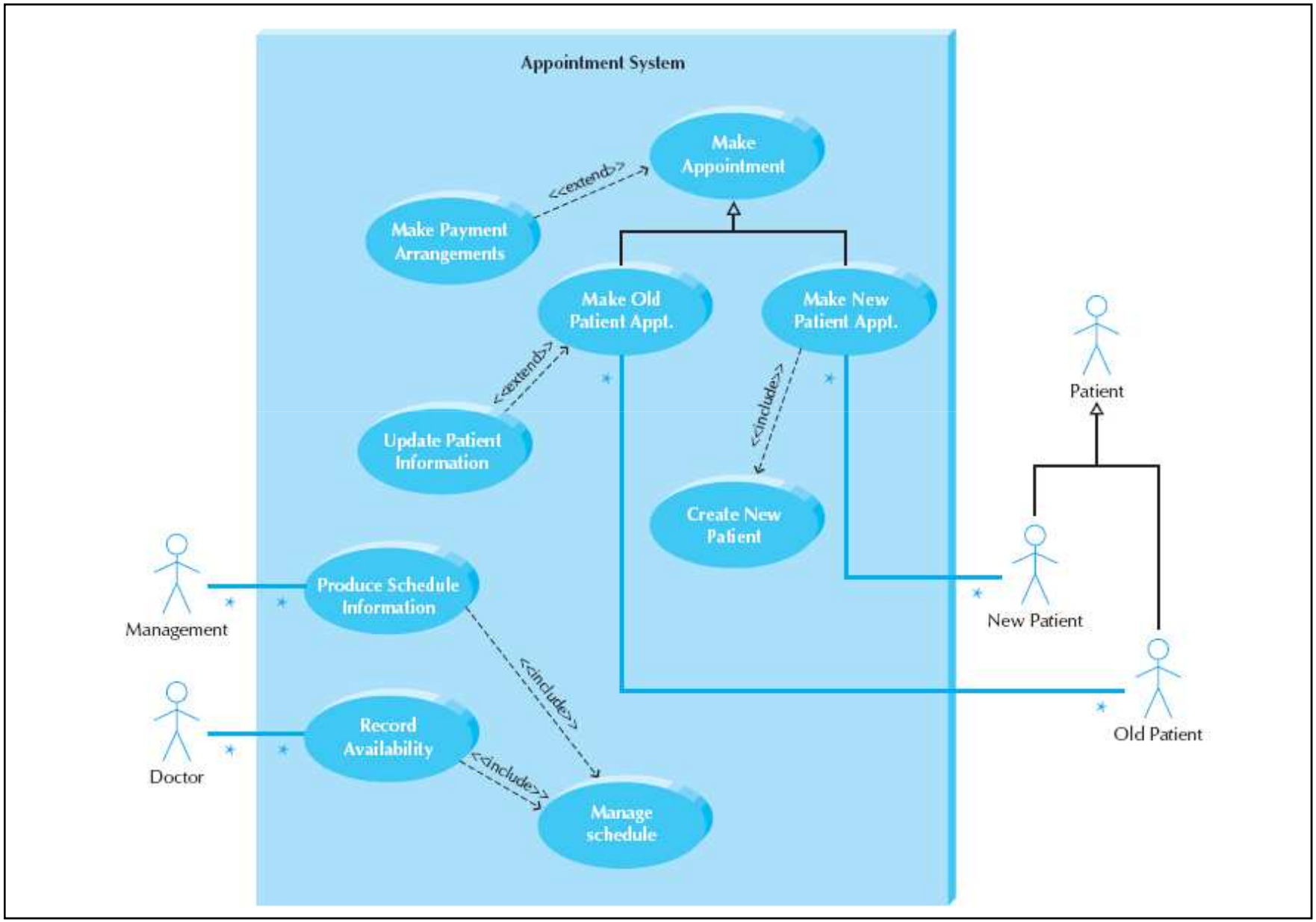
Use-Case Diagram with Specialized Actor



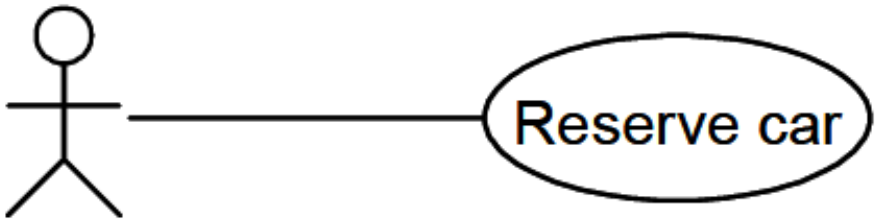
What are Use-Case Descriptions?

- Describe basic functions of the system using words
  - What the user can do
  - How the system responds
- Use Case description overview
  - Name
  - ID Number
  - Type
  - Primary Actor
  - Brief Description
  - Importance Level
  - Stakeholder(s)
  - Trigger(s)

Sample Use Case



A Complete Example of a Use Case 1/3



Use case: Reserve car
Scope: EU-Rent system
Primary Actor: Customer
Stakeholders and interests:
Customer: Wants to make a car rental reservation.
Branch manager: Wants to ensure that the reservation can be honored.
Company: Rentals requested by blacklisted customers must be refused.
Company: Wants to ensure that customers receive the best price for their rental.
Precondition: None.
Success Guarantees: Reservation is saved. Reservation can be honored. Price is correctly calculated.

A Complete Example of a Use Case 2/3

Trigger: The customer wants to make a rental reservation for a car.
Main Success Scenario:
1. The customer identifies himself.
2. The system verifies that the customer has not been blacklisted.
3. The customer describes the rental reservation he wants to make by specifying the rental period, the pickup branch, the drop-off branch, and the car group.
4. The system verifies that the customer is allowed to make the reservation.
5. The system verifies that there may be cars available in the desired car group for the duration of the rental.
6. The system presents the price of the rental.
7. The customer accepts the rental proposal.
8. The system saves the reservation.
9. The system confirms the rental reservation to the customer.

A Complete Example of a Use Case 3/3

Extensions:

1a. The customer is new:

1a1. Create customer.

2a. The customer has been blacklisted:

2a1. The system notifies customer. Use case ends.

4a. The customer is not allowed to make the reservation:

4a1. The system notifies the customer.

4a2. The customer changes the rental period.

4a2a. The customer decides to exit:

4a2a1. Use case ends.

5a. There are no cars available:

5a1. The system notifies the customer.

5a2. The customer changes the car group or the rental period.

5a2a. The customer decides to exit:

5a2a1. Use case ends.

7a. The customer refuses the proposal:

7a1. The customer changes the car group or the rental period.

7a1a. The customer decides to exit:

7a1a1. Use case ends.

7b. The customer wants to guarantee the rental:

7b1. The customer gives his credit card information.

Major Steps in Writing Use-Cases

- Identify the major use-cases
- Create the use-case diagram
- Expand the major use-cases
- Confirm the major use-cases
- Write Use case descriptions

Use-Case Points

- A size and effort estimation technique that was developed around use cases
- Requires at a minimum:
  - The set of essential use cases
  - The use case diagram
  - All actors and use cases classified as simple, average, or complex

Unadjusted Actor Weighting Table:

Actor Type	Description	Weighting Factor	Number	Result
Simple	External System with well-defined API	1		
Average	External System using a protocol-based interface, e.g., HTTP, TCT/IP, or a database	2		
Complex	Human	3		
		Unadjusted Actor Weight Total (UAW)		

Unadjusted Use Case Weighting Table:

Use-Case Type	Description	Weighting Factor	Number	Result
Simple	1–3 transactions	5		
Average	4–7 transactions	10		
Complex	>7 transactions	15		
		Unadjusted Use-Case Weight Total (UUCW)		

Unadjusted Use Case Points (UUCP) = UAW + UUCW

Technical Complexity Factors:

Factor Number	Description	Weight	Assigned Value (0–5)	Weighted Value	Notes
T1	Distributed system	2.0			
T2	Response time or throughput performance objectives	1.0			
T3	End-user online efficiency	1.0			
T4	Complex internal processing	1.0			
T5	Reusability of code	1.0			
T6	Easy to install	0.5			
T7	Ease of use	0.5			
T8	Portability	2.0			
T9	Ease of change	1.0			
T10	Concurrency	1.0			
T11	Special security objectives included	1.0			
T12	Direct access for third parties	1.0			
T13	Special user training required	1.0			
		Technical Factor Value (TFactor)			

Technical Complexity Factor (TCF) = 0.6 + (0.01 * TFactor)

Environmental Factors:

Factor Number	Description	Weight	Assigned Value (0 – 5)	Weighted Value	Notes
E1	Familiarity with system development process being used	1.5			
E2	Application experience	0.5			
E3	Object-oriented experience	1.0			
E4	Lead analyst capability	0.5			
E5	Motivation	1.0			
E6	Requirements stability	2.0			
E7	Part time staff	–1.0			
E8	Difficulty of programming language	–1.0			
		Environmental Factor Value (EFactor)			

Environmental Factor (EF) = 1.4 + (−0.03 * EFactor)
Adjusted Use Case Points (UCP) = UUCP * TCF * ECF
Effort in Person Hours = UCP * PHM

Actor & Use Case Weighting Tables

Unadjusted Actor Weighting (**UAW**)

Actor Type	Description	Weighting Factor
Simple	External System with well-defined API	1
Average	External System using a protocol-based 2 interface, e.g., HTTP, TCT/IP, or a database	2
Complex	Human	3

Unadjusted Use Case Weighting (**UUCW**)

Use-Case Type	Description	Weighting Factor
Simple	1-3 transactions	5
Average	4-7 transactions	10
Complex	More than 7 transactions	15

Unadjusted Use Case Points (UUCP) = UAW + UUCW

Technical Complexity Factors

Factor Number	Description	Weight
T1	Distributed system	2.0
T2	Response time or throughput performance objectives	1.0
T3	End-user online efficiency	1.0
T4	Complex internal processing	1.0
T5	Reusability of code	1.0
T6	Easy to install	0.5
T7	Ease of use	0.5
T8	Portability	2.0
T9	Ease of change	1.0

Technical Complexity Factor (TCF) = 0.6 + (0.01 * TFactor)

Environmental Factors

Factor Number	Description	Weight
E1	Familiarity with system development process in use	1.5
E2	Application experience	0.5
E3	Object-oriented experience	1.0
E4	Lead analyst capability	0.5
E5	Motivation	1.0
E6	Requirements stability	2.0
E7	Part time staff	-1.0
E8	Difficulty of programming language	-1.0

Environmental Factor (EF) = 1.4 + (-0.03 * EFactor)

Person-Hours Multiplier

If the sum of (number of Efactors E1 through E6 assigned value < 3) and (number of Efactors E7 and E8 assigned value > 3) ≤ 2

PHM = 20

Else If the sum of (number of Efactors E1 through E6 assigned value < 3) and (number of Efactors E7 and E8 assigned value > 3) = 3 or 4

PHM 28

Else

Rethink project; it has too high of a risk for failure

Computing Use-Case Points

- Adjusted Use Case Points (UCP) =
  UUCP * TCF * EF
- Effort in Person Hours =
  UCP * PHM

Thank you!

Technical Factor		Multiplier	Relative Magnitude (Enter 0-5)	Description
1	**Distributed System Required**	2		The architecture of the solution may be centralized or single-tenant , or it may be distributed (like an n-tier solution) or multi-tenant. Higher numbers represent a more complex architecture.
2	**Response Time Is Important**	1		The quickness of response for users is an important (and non-trivial) factor. For example, if the server load is expected to be very low, this may be a trivial factor. Higher numbers represent increasing importance of response time (a search engine would have a high number, a daily news aggregator would have a low number).
3	**End User Efficiency**	1		Is the application being developed to optimize on user efficiency, or just capability? Higher numbers represent projects that rely more heavily on the application to improve user efficiency.
4	**Complex Internal Processing Required**	1		Is there a lot of difficult algorithmic work to do and test? Complex algorithms (resource leveling, time-domain systems analysis, OLAP cubes) have higher numbers. Simple database queries would have low numbers.
5	**Reusable Code Must Be A Focus**	1		Is heavy code reuse an objective or goal? Code reuse reduces the amount of effort required to deploy a project. It also reduces the amount of time required to debug a project. A shared library function can be re-used multiple times, and fixing the code in one place can resolve multiple bugs. The higher the level of re-use, the lower the number.
6	**Installation Ease**	0.5		Is ease of installation for end users a key factor? The higher the level of competence of the users, the lower the number.
7	**Usability**	0.5		Is ease of use a primary criteria for acceptance? The greater the importance of usability, the higher the number.
8	**Cross-Platform Support**	2		Is multi-platform support required? The more platforms that have to be supported (this could be browser versions, mobile devices, etc. or Windows/OSX/Unix), the higher the value.
9	**Easy To Change**	1		Does the customer require the ability to change or customize the application in the future? The more change / customization that is required in the future, the higher the value.
10	**Highly Concurrent**	1		Will you have to address database locking and other concurrency issues? The more attention you have to spend to resolving conflicts in the data or application, the higher the value.
11	**Custom Security**	1		Can existing security solutions be leveraged, or must custom code be developed? The more custom security work you have to do (field level, page level, or role based security, for example), the higher the value.
12	**Dependence On Third-Party Code**	1		Will the application require the use of third party controls or libraries? Like re-usable code, third party code can reduce the effort required to deploy a solution. The more third party code (and the more reliable the third party code), the lower the number.
13	**User Training**	1		How much user training is required? Is the application complex, or supporting complex activities? The longer it takes users to cross the suck threshold (achieve a level of mastery of the product), the higher the value.
Calculated TCF			**0.6**	

Environmental Factor		Multiplier	Relative Magnitude (Enter 0-5)	Description
1	**Familiarity With The Project**	1.5		How much experience does your team have working in this domain? The domain of the project will be a reflection of what the software is intended to accomplish, not the implementation language. In other words, for an insurance compensation system written in java, you care about the team's experience in the insurance compensation space - not how much java they've written. Higher levels of experience get a higher number.
2	**Application Experience**	0.5		How much experience does your team have with the application. This will only be relevant when making changes to an existing application. Higher numbers represent more experience. For a new application, everyone's experience will be 0.
3	**OO Programming Experience**	1		How much experience does your team have at OO? It can be easy to forget that many people have no object oriented programming experience if you are used to having it. A user-centric or use-case-driven project will have an inherently OO structure in the implementation. Higher numbers represent more OO experience.
4	**Lead Analyst Capability**	0.5		How knowledgeable and capable is the person responsible for the requirements? Bad requirements are the number one killer of projects - the Standish Group reports that 40% to 60% of defects come from bad requirements. Higher numbers represent increased skill and knowledge.
5	**Motivation**	1		How motivated is your team? Higher numbers represent more motivation.
6	**Stable Requirements**	2		Changes in requirements can cause increases in work. The way to avoid this is by planning for change and instituting a timing system for managing those changes. Most people don't do this, and some rework will be unavoidable. Higher numbers represent more change (or a less effective system for managing change).
7	**Part Time Staff**	-1		Note, the multiplier for this number is negative. Higher numbers reflect team members that are part time, outside consultants, and developers who are splitting their time across projects. Context switching and other intangible factors make these team members less efficient.
8	**Difficult Programming Language**	-1		This multiplier is also negative. Harder languages represent higher numbers. We believe that difficulty is in the eye of the be-coder (groan). Java might be difficult for a fortran programmer. Think of it in terms of difficulty for your team, not abstract difficulty.
Calculated EF			**1.4**	