Chapter 7
Solutions of Problems

7.1 Chapter 1

Problem 1.6.1: Data, information, and knowledge
A customer calls the requirements engineer and tells her about a feature they forgot to put into the specification. Where are data, information, and knowledge in this example?

The missing characters in the specification constitute "data." There is also "audio data" of the customer's voice. "Information" implies meaning. Interpretation of characters or spoken words as describing a feature of the product turns the data into "information." Both requirements engineer and customer have additional "knowledge" about the system before they start the conversation. This context turns the information about a new feature into "knowledge" that is related to other knowledge in their minds.

Problem 1.6.2: Missing the expert
Explain what could happen if a customer cannot reach the requirements engineer, but reaches a sales person instead. Assume the requirements engineer knows the project very well.

Data in terms of spoken words is transmitted just as in the previous problem. However, because of a lack of background knowledge, the sales person may be unable to interpret those words: Information about the missing feature could be lost when the sales person misinterprets what is being said. By all means, the sales person is missing contextual *knowledge*, so the *information* about the missing feature cannot be contextualized within preexisting *knowledge*. In this example, *data* has been transferred, *information* may be partially transmitted, but *knowledge* has not been successfully transferred.

Problem 1.6.3: Exact meaning
Your company has a cooperation project with an offshore development organization. You are not sure whether they use the same version of UML as you do. How do you make sure to transfer not just data but also the information implied in your UML 2.0 diagrams? Discuss your attempt using the terminology introduced in Chap. 1.

DOI 10.1007/978-3-540-95880-2_7, © Springer-Verlag Berlin Heidelberg 2009

UML 2.0 is formally defined. When a UML 2.0 diagram is used to represent information, it is important to let the receiver know about this formal definition. For that purpose, a piece of metadata must be conveyed together with the operational data (the UML diagram itself). This metadata must at least tell the receiver that you used UML 2.0. Maybe, there should be additional metadata on the implications of that fact. After reading the metadata, receivers can interpret the diagram as UML 2.0 formally defined data. This helps them in assigning semantics (the implied meaning) to the UML syntax. In summary, the operational data (the diagram) is transferred together with metadata pointing out the version of UML and the formal meaning of the symbols. As a consequence, semantics is received together with syntax, thus conveying all information expressed in the details of a UML 2.0 diagram. With a knowledgeable receiver, this information can be reconstructed into the knowledge the sender intended to express in the UML diagram. Without that metadata, a receiver might have treated the diagram as a less formally defined sketch and missed part of the meaning (i.e., information and knowledge).

Problem 1.6.4: Experience in a review

During a review of a design document, the team finds out that there was a misunderstanding among customer representatives: They did not really need a distributed system. As a consequence, most of the design effort invested was wasted. Two of the authors participate in the review, and one will be told about it later. What will be the experiences of the authors and reviewers? Describe their observations, their emotions, and possible conclusions they may draw. Emphasize differences between different authors and between authors and reviewers.

Participating authors: They have observed the review and the finding. They "observed" the conversation that uncovered the misunderstanding. As they have invested so much effort into a document by mistake, they will be frustrated or angry – a strong negative emotion. They might draw very different conclusions from this observation and emotion: Either they might conclude that "it is not worth putting that much effort into a document, as it might be wasted effort" or they might conclude to "never start writing the full-fledged document without checking basic assumptions." It is important to note that there is no single, objective, or obvious conclusion in an observation – but any conclusion drawn will have an impact on future actions and performance.

The author who did not participate in the review does not share the same observation and, hence, misses the first-hand emotion. When the other authors tell him what happened, their own conclusions will influence their report. The listening author will "observe" this report, which will cause a second-hand emotion and – probably – a similar conclusion.

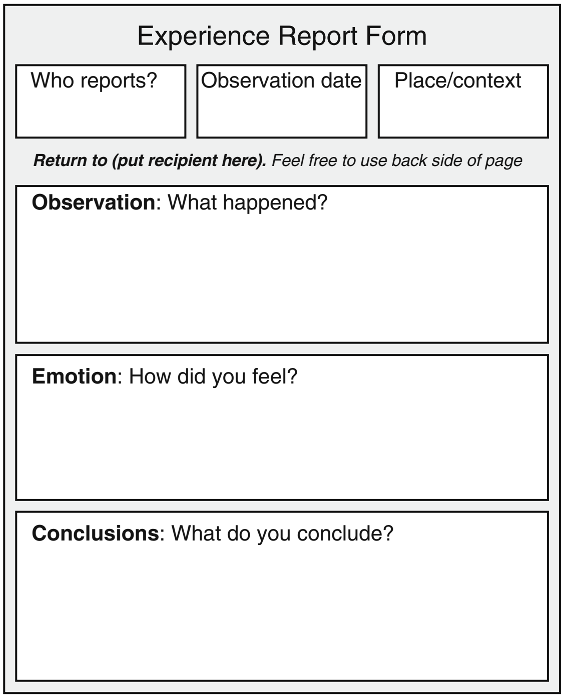
Reviewers participated in the same situation but see it from a different perspective. Their observation will include the success of finding a severe misunderstanding. Although this will cause extra effort for the designers, the reviewers will feel good about having detected the problem. Implementation effort was not wasted. Reviewers' emotions could be positive; at worst, they might feel bad for the authors. The resulting conclusion will look very different from the authors'. "A review is always worth the time, as you may find hidden misunderstandings" is one, but the author conclusion "never start writing the full-fledged document without checking basic assumptions" may also be concluded by the reviewers. This exercise highlights the importance of experiences – even though they may be highly individual, different, and unpredictable.

Problem 1.6.5: Experience capture form

Sketch a one-page form for capturing experiences when they occur. Explain your design and discuss how you will be able to effectively and efficiently collect what you need for your form and how you will use the collected information later. Did your first sketch cover all relevant information for successful reuse?

A predefined experience report form should help people to remember what needs to be recorded about a memorizable observation. It should solicit experiences and support people in documenting them – but it should not restrict their desire to express themselves. Therefore, experience collection mechanisms need to offer guidance without limiting expressiveness.

The sketch below is just one example. It contains only the essential fields (guidance) but allows people to add more or different information on the back. Note the "return to . . ." tag. Each report form should show where it needs to be sent when completed: by fax, e-mail, or company mail.



Experience Report Form

Who reports?	Observation date	Place/context

Return to (put recipient here). *Feel free to use back side of page*

Observation: What happened?

Emotion: How did you feel?

Conclusions: What do you conclude?

The small fields collect essential metadata:

- Who made the observation?
- When was it observed (not documented!)?
- In which context was it collected?

Most space is reserved for operational data about the three key components of an experience:

- What was observed?
- What was the emotion it provoked?
- What is the conclusion or hypothesis derived from that?

The page should allow ample room for the operational data and allow people to document the experience with the lowest effort possible. It is rarely advisable to ask for more details or aspects. People who want to tell more can do so on the reverse side. People in a hurry will be turned off by more questions.

7.2 Chapter 2

Problem 2.7.1: Single- and double-loop learning
What is the difference between single-loop and double-loop learning? Explain both modes and give a short example from software engineering for each of the two.

In single-loop learning, a given goal is to be reached. Learning leads to reaching the goal faster or better. For example, a skilled software designer will become faster in drawing UML diagrams through single-loop learning. In double-loop learning, however, setting, reflecting, and adjusting goals are included in the learning process. In the software engineering example, software engineers will learn to use UML diagrams in only those situations in which they are advantageous. They may use Petrinets in other situations (e.g., for specifying process synchronization).

Problem 2.7.2: Framework XY learning scenario
A software company has used a certain framework XY for building business applications. Problems using that framework are reported to a hotline. The hotline releases a new version of its newsletter "XY Procedures." New projects are supposed to follow those procedures. This is supposed to spread learning across all projects. What kind of learning is this? How could this type of learning turn out to be counterproductive? Refer to the XY example. What concrete activities could help to prevent that negative effect?

The newsletter promotes single-loop learning: Existing procedures are presented and are supposed to be followed without criticizing them. However, if the XY framework turns out to be inadequate for a project, following the procedures might add to the problem: A better framework should be proposed instead, thus adjusting the goal. Promoted procedures could also be modified in order to circumvent the disadvantages of framework XY in the given project. In all cases, single-loop learning of using XY in the standard way is not helpful but counterproductive.

Problem 2.7.3: Formal reasoning patterns
You are working as a test engineer. Over the years, you have noticed that many tests fail due to the incomplete initialization of variables. Describe the three formal reasoning patterns (abduction, deduction, induction) and label the following statements with the corresponding reasoning pattern name:

Induction: A general rule is derived from one or more observations. This reasoning may be flawed.

Deduction: A special case is derived by applying a general rule.

Abduction: From an observation in a special case, a hypothesis is derived (what led to the observed situation?) as well as a general principle that supports the hypothesis.

Answers to the examples:

- "...there are often errors in initialization."

  Induction: Some errors were observed, the general rule ("are often") is derived.
- "Initialization looks so trivial, so many people are not very careful about it."

  Abduction: Provides an explanation for a specific observation (people forget initializations). Hypothesis leads to general rule ("people are not careful when they consider something trivial" and: "initialization looks trivial").
- "Programmers make more and more mistakes."

  Induction: General rule derived from several observations.
- "Setting a counter to 0 or 1 is often forgotten."

  Deduction from the general rule derived above ("there are often errors in initialization"). This is a special case of an initialization: setting a counter to 0 or 1.
- "Flawed initialization is easy to find by testing; that is why we find so much, because we test more systematically."

  Abduction: An observation is explained by a hypothesis about a generally applicable principle. Like induction, a principle is derived from a few (or a single) observations.
- "Programs have errors."

  Induction: The programs we have seen had errors, so we derive a general rule: (all) programs (seem to) have errors.

Problem 2.7.4: Schools and approaches of knowledge management
A company wants to encourage the exchange of knowledge among their software engineers. For each of the following suggestions, identify the respective school (according to Earl) and approach (product- or process-centric) it belongs to:

- "We develop a knowledge database and send an e-mail to all software engineers to enter their knowledge."

  Technocratic/Systems, product-centric: Infrastructure perspective.

- "Let's put an info terminal in the main lobby; every time you enter the building, you will see a new 'knowledge item of the day'."

  Behavioral/Space. Process-centric: The daily learning impulse.
- "We could use a Wiki to record everything we think others might need."

  Technocratic/Systems. Product-centric: A mechanism for exchanging knowledge chunks.
- "Okay, but there needs to be a moderator; plus, we should have monthly meetings of the Wiki User Group."

  Behavioral/Organizational. Process-centric: The learning aspect in several facets is being addressed.
- "Let's put up a coffee machine and two sofas."

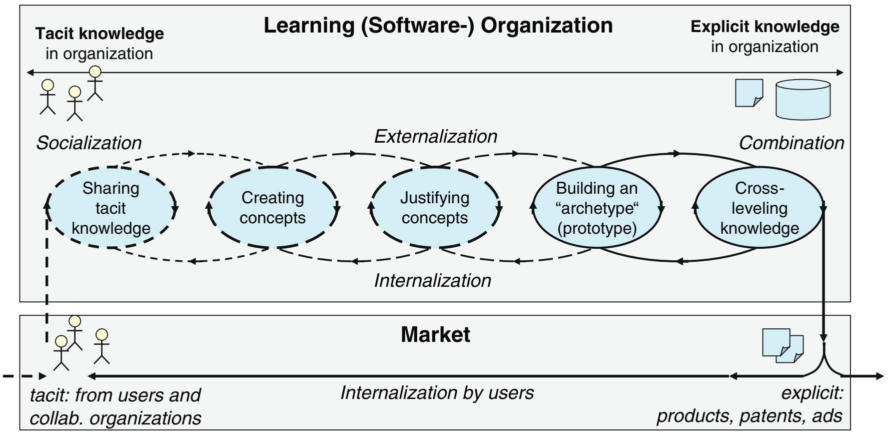
  Behavioral/Spatial. Process-centric: Psychological and social aspects are highlighted, logistics of knowledge are not addressed.
- "There are powerful networks now, so we can even send movies. Everybody gets a camera on their desk, and when they want or have something interesting, they record a movie."

  Technocratic/Engineering. Product-centric: Movies are chunks of "knowledge" or "question," and this statement is concerned with the logistics of those movies.
- "Great idea! We hire a person who can index all incoming movies according to their SWEBOK category, and a few other attributes. We build a resolution machine that helps to match new entries with stored ones."

  Technocratic/Cartographic. Product-centric: Even more obvious focus on storing and managing chunks of "knowledge."

Problem 2.7.5: Knowledge management life-cycle

Draw Nonaka and Takeuchi's knowledge management life-cycle. Explain it by applying the concepts to the example of a group of software project leaders who meet in a community of practice in order to learn about cost estimation.


Learning (Software-) Organization
Tacit knowledge in organization
Explicit knowledge in organization
Socialization
Externalization
Combination
Sharing tacit knowledge
Creating concepts
Justifying concepts
Building an "archetype" (prototype)
Cross-leveling knowledge
Internalization
Market
tacit: from users and collab. organizations
Internalization by users
explicit: products, patents, ads

The CoP meets to exchange experience and knowledge. During the first meetings, most knowledge is implicit and often tacit. Participants will learn by socializing and talking about their projects. Some may not even be able to externalize their (tacit!) knowledge on cost estimation; they simply "do it." The CoP encourages its members to enter into the next phase of the life cycle: Those who listen might be able to derive concepts (by induction or abduction). Those concepts need to be tested, so later meetings can be used to collect additional reports in order to validate the concepts. During this phase, the more experienced participants externalize what they know (adding to the concepts, and finally, a constructive prototype of a cost estimation model). Others learn by internalizing the stories they hear: Applying models helps them to deeply understand (and internalize) what they hear. The final product of the cycle will be an externalized, documented package including the elicited and externalized knowledge that has been combined into a prototype, and finally a model with variants. This model is disseminated in the organization (in the CoP, to start with). When new needs arise to update or correct the model, the CoP may enter into a new iteration of the cycle.

7.3 Chapter 3

Problem 3.8.1: Pattern
What is a "pattern" with respect to knowledge and experience management? What are the core elements of a pattern and what are they used for when describing and reusing software engineering knowledge? Give an example from software testing.

A pattern is defined as a situation and a consequence or a problem (situation) and a solution (consequence). Core elements are the application condition, the situation or problem that is characteristic of the pattern, and the consequence. The consequence is usually the solution proposed by the pattern. Experience and knowledge can be represented in patterns. By making the application condition (including situation and context) explicit, this knowledge can be used more easily.

Example: A testing expert has seen many cases in which only very few errors were found during the first 3 days of testing. According to his experience, this is not an indication of supreme software quality but rather points to a poor testing strategy. This experience can be represented as:

- Problem/situation: Very few errors found during first 3 days of testing.
- Solution: Review testing strategy.

Although those two aspects make a pattern, it will be advantageous to document the rationale for the solution. This will often point to observations and experiences, like the concrete observations made in this example. Whenever the situation occurs in a project, the pattern should be used, and poor testing strategies will be detected. Applying the pattern is simple, and the benefit may be significant: No project will be deceived by the low number of findings. Even if the pattern should be wrong every now and then, this is not a major problem.

Note that the “problem” does not look like a problem at first. Therefore, it is more appropriate to call the triggering condition the “application situation.”

Problem 3.8.2: Defining quality aspects

Finding a common language with the customer is important in software engineering. Assume you are defining quality attributes for a new piece of software with a customer from a medical background. Why is it important to define key terms like “reliability” or “ease of use” explicitly?

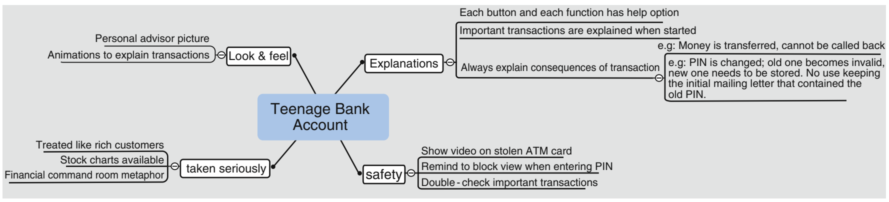
Customers and other project participants may use the same terms in different contexts and with different meanings (homonyms). In most cases, meanings will be similar but differ in important details. Quality attributes are part of quality requirements. Meeting the (quality) requirements is an important goal of a project, so it should be clear what they really mean. A misunderstanding could turn into a major problem: A medical customer may associate the reliability of a device with its availability during surgeries. A computer person may think about correctness of operation. Both associations of reliability are not precisely conformant to quality standards, but both “definitions” of reliability can be found in real projects. The common language needs to bridge the gap between participants.

Assume you are developing a banking system for teenagers on the Internet. This program is supposed to target young, inexperienced banking customers. They should be offered basic and additional information on the banking products, and each of their interactions should be explained in detail if they wish. Also, teenage customers should not be allowed to overdraw their account. This example is used for the following problems.

Problem 3.8.3: Mind map

In the process of developing the system, innovative ideas and important reminders are collected in a mind map. The intention is to gain an overview of important and “cool and catchy” concepts that should be taken into account to make the teenage bank account system a success. Draw a mind map of a teenage bank account with at least four directly related concepts and about two to four comments each. Comments should explain the innovation or importance of each concept.

(See the accompanying mind map.)


Teenage Bank Account
Look & feel
Personal advisor picture
Animations to explain transactions
Explanations
Each button and each function has help option
Important transactions are explained when started
Always explain consequences of transaction
e.g: Money is transferred, cannot be called back
e.g: PIN is changed; old one becomes invalid, new one needs to be stored. No use keeping the initial mailing letter that contained the old PIN.
taken seriously
Treated like rich customers
Stock charts available
Financial command room metaphor
safety
Show video on stolen ATM card
Remind to block view when entering PIN
Double - check important transactions

Problem 3.8.4: Brainstorming versus mind map
When you drew the mind map, you were not performing brainstorming, although some people might call it so. What are the main differences between drawing your mind map and "real" brainstorming?

Here are some important deviations. Not all of them may apply in your situation:

- Drawing the map alone. Brainstorming is supposed to be carried out in a group; participants should base their contributions on what they hear and see from others.
- In particular, a moderator is missing who would observe the brainstorming rules. In many cases, the moderator will also write down the contributions.
- Associations and relationships in a mind map were drawn while the mind map was generated. In brainstorming, the moderator should not influence the process, and there should be no online structuring (such as relationships, labeling, etc.).

Problem 3.8.5: Glossary entries
Provide glossary entries for "graphical user interface," "bank account," and "ATM card" (no debt allowed for teenagers) with respect to the teenage bank account.

- **Graphical user interface**: The screen of the banking application consists of different graphical elements for input and output. Young customers interact with the banking application by interacting with those elements, similar to a computer game. For example, screens contain symbols that represent functions. They can be clicked to trigger the respective function. Dragging and dropping banknote symbols with the mouse indicates money transfer (and so on: it is important to refer to specific aspects of graphical user interfaces in this application).
- **Bank account**: The amount of money put into the bank is stored individually for each customer. Administrative information (such as name, age, address, etc.) is stored together with the monetary information. When managed by the bank, the set of personal and monetary information is called an "account." It is identified by an eight-digit "account number," which is also part of the account. One customer may have multiple accounts, which will have different account numbers. (It is important to explain the term so that young customers will understand it. Most of them are computer-literate but not used to banks and financial operations.)
- **ATM card**: Personal card that enables a customer to carry out banking operations at a banking machine (called ATM; Automated Teller Machine). Customers receive ATM cards at their request. They cannot withdraw more money than they have in the bank. (This entry could have more or less detail on how to use the ATM card. In principle, a short definition might be enough. It is not recommended to explain the rules of usage within a glossary entry.)

Problem 3.8.6: Typos
Typos (incorrectly spelled words) are more common in mind maps than in glossaries. Why is that so and why is it usually not a problem for mind maps?

- Mind maps are mostly drawn using a computerized tool. The person drawing and writing tries to capture contributions of the entire group and write them down fast. This process requires drawing lines and links, rephrasing what people say, and typing fast. A typo will mostly not endanger meaning, so little care will be taken in deleting small mistakes.
- Glossaries will be used as a reference in a project. They should not have obvious mistakes, as those might put their credibility at stake. For that reason, a glossary will be checked before it is used in a project. In extreme cases, a typo or mistake may ripple through many documents that use the flawed glossary entry.

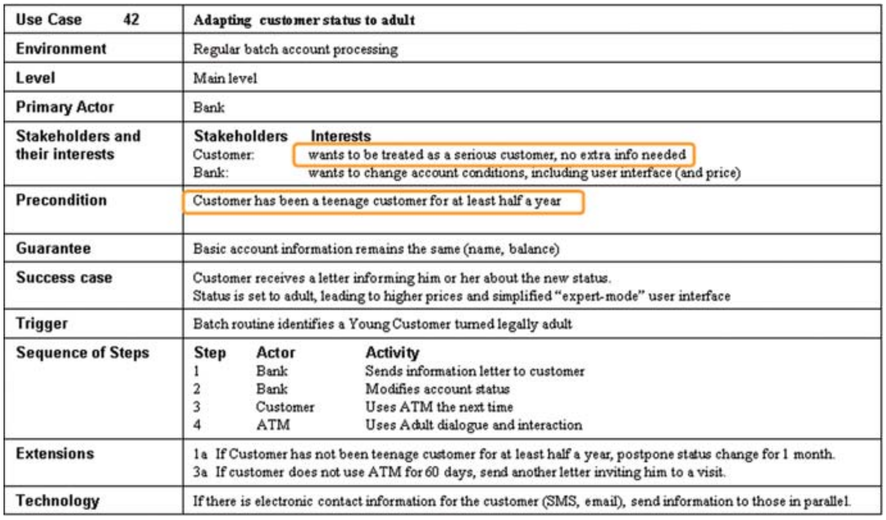
Problem 3.8.7: Domain model

What happens when a teenage customer turns legally adult? How can you find out what is supposed to happen then? Write a use case for this operation and highlight two pieces of "domain knowledge" it contains.

From the above-mentioned description, it is not clear what will happen. This situation of a customer turning adult may have been forgotten.

In that case, the customer (i.e., the bank building the young customer portal) will need to decide. A use case can help to facilitate precise communication between developing organization and bank. The following example shows a process that may be carried out automatically or manually. The two pieces highlighted describe simple but important pieces of domain knowledge that have been made explicit using the use case.

(See Use Case 42, "Adopting customer status to adult," below.)

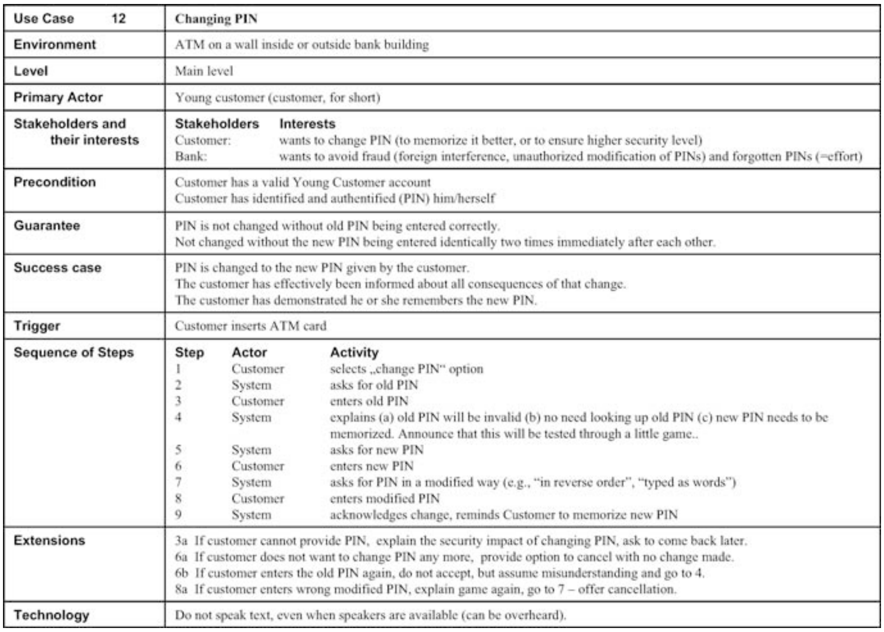


Use Case 42	**Adapting customer status to adult**
Environment	Regular batch account processing
Level	Main level
Primary Actor	Bank
Stakeholders and their interests	**Stakeholders** **Interests** Customer: wants to be treated as a serious customer, no extra info needed Bank: wants to change account conditions, including user interface (and price)
Precondition	Customer has been a teenage customer for at least half a year
Guarantee	Basic account information remains the same (name, balance)
Success case	Customer receives a letter informing him or her about the new status. Status is set to adult, leading to higher prices and simplified "expert-mode" user interface
Trigger	Batch routine identifies a Young Customer turned legally adult
Sequence of Steps	**Step** **Actor** **Activity** 1 Bank Sends information letter to customer 2 Bank Modifies account status 3 Customer Uses ATM the next time 4 ATM Uses Adult dialogue and interaction
Extensions	1a If Customer has not been teenage customer for at least half a year, postpone status change for 1 month. 3a If customer does not use ATM for 60 days, send another letter inviting him to a visit.
Technology	If there is electronic contact information for the customer (SMS, email), send information to those in parallel.

Problem 3.8.8: Use case

Describe the use case of "changing the PIN" using the use-case template shown in Fig. 3.8. Make sure to address the characteristic need of young customers to learn about permitted PIN formats and the implications of changing a PIN (and maybe forgetting it). What happens when they enter incorrect input?

When they enter incorrect input, additional information is displayed – in contrast with normal ATM software that simply requests the correct PIN again.
(See Use Case 12, "ChangingPIN," below.)



Use Case 12	**Changing PIN**
Environment	ATM on a wall inside or outside bank building
Level	Main level
Primary Actor	Young customer (customer, for short)
Stakeholders and their interests	**Stakeholders** **Interests** Customer: wants to change PIN (to memorize it better, or to ensure higher security level) Bank: wants to avoid fraud (foreign interference, unauthorized modification of PINs) and forgotten PINs (=effort)
Precondition	Customer has a valid Young Customer account Customer has identified and authentified (PIN) him/herself
Guarantee	PIN is not changed without old PIN being entered correctly. Not changed without the new PIN being entered identically two times immediately after each other.
Success case	PIN is changed to the new PIN given by the customer. The customer has effectively been informed about all consequences of that change. The customer has demonstrated he or she remembers the new PIN.
Trigger	Customer inserts ATM card
Sequence of Steps	**Step** **Actor** **Activity** 1 Customer selects „change PIN" option 2 System asks for old PIN 3 Customer enters old PIN 4 System explains (a) old PIN will be invalid (b) no need looking up old PIN (c) new PIN needs to be memorized. Announce that this will be tested through a little game.. 5 System asks for new PIN 6 Customer enters new PIN 7 System asks for PIN in a modified way (e.g., "in reverse order", "typed as words") 8 Customer enters modified PIN 9 System acknowledges change, reminds Customer to memorize new PIN
Extensions	3a If customer cannot provide PIN, explain the security impact of changing PIN, ask to come back later. 6a If customer does not want to change PIN any more, provide option to cancel with no change made. 6b If customer enters the old PIN again, do not accept, but assume misunderstanding and go to 4. 8a If customer enters wrong modified PIN, explain game again, go to 7 – offer cancellation.
Technology	Do not speak text, even when speakers are available (can be overheard).

Problem 3.8.9: Writing a pattern

Let us assume the company developing the teenage banking system has gathered experience with many other systems for young people. During those projects, there was a recurring misunderstanding: When young customers were interviewed for requirements, they rarely checked "intuitive interface" as a high priority. Nevertheless, customer satisfaction seemed to depend on ease of use. Structure this observation as a pattern, and describe how the teenage banking project can make use of it.

Observation as a pattern:

- Situation/Problem: A young customer project.
- Solution: Ease of use is an important quality aspect. This is true no matter what teenage customers say in interviews.

Different possibility: We conclude from the observation that young customers either did not understand the term "intuitive interface" or they did not consider it important. This conclusion would lead to a different pattern:

- Situation: Interviewing teenage customers about software quality requirements.
- Solution: Ask about the importance of intuitive interface by using examples and concrete prototypes.
- Rationale: Teenagers seem to be unable or unwilling to associate with the abstract term of "intuitive interface." The question must be asked in a way teenagers can relate to.

The teenage banking project can benefit from either of the two patterns: When they find the situation matches their own situation, the solution part can be treated as advice. In the two examples: (1) ease of use will be treated as important, no matter what the interviewees say, and (2) the interviews will consider the teenagers' tendency to ignore or underestimate interface issues. Examples and concrete scenarios will help.

7.4 Chapter 4

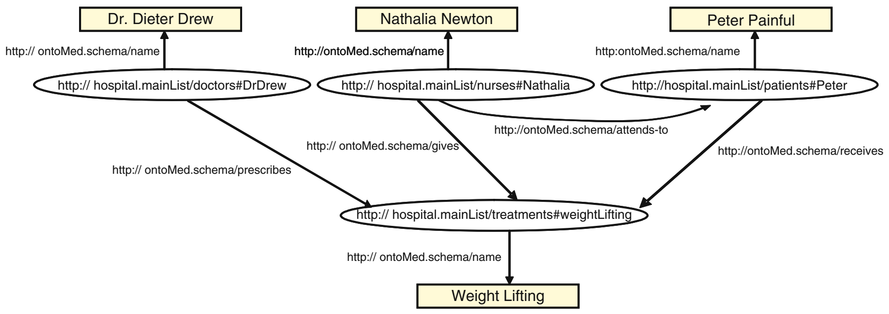
Problem 4.9.1: Definition and purpose
What is the definition and the purpose of an ontology?

An ontology is a data model that represents a set of concepts within a domain and the relationships between those concepts. It is used to reason about the objects within that domain (Wikipedia, August 30, 2007).

An ontology provides a clear reference for a certain domain. Knowledge workers, tools, and projects can refer to it. By introducing formally defined concepts, relationships, and properties, those elements can be used in searches, reasoning, and for computer-supported tasks.

Problem 4.9.2: RDF graph
Use the ontology sketched in Fig. 4.1: Describe a situation as an RDF graph in which Dr. Dieter Drew prescribes "weight lifting" as a physical therapy to patient Peter Painful. Nathalia Newton is a nurse who assists Peter in doing the weight lifting in a health-stimulating way. Treat names as string attributes defined in that same ontology.

Use http://ontoMed.schema/ as the name for the ontology shown in Fig. 4.1. The hospital maintains a list on the intranet. Employees, patients, and treatments offered are listed in http://hospital.mainList, with the subcategories /doctors, /nurses, /patients, and /treatments. A specific entry (e.g., XY) is accessible at #XY.


Dr. Dieter Drew
Nathalia Newton
Peter Painful
http:// ontoMed.schema/name
http://ontoMed.schema/name
http:ontoMed.schema/name
http:// hospital.mainList/doctors#DrDrew
http:// hospital.mainList/nurses#Nathalia
http://hospital.mainList/patients#Peter
http://ontoMed.schema/attends-to
http:// ontoMed.schema/gives
http://ontoMed.schema/receives
http:// ontoMed.schema/prescribes
http:// hospital.mainList/treatments#weightLifting
http:// ontoMed.schema/name
Weight Lifting

Problem 4.9.3: Inheritance

Explain what happened to the inheritance relationships in Fig. 4.1 when you drew the RDF graph, and why.

Inheritance relationships are not visible in the RDF graph. The RDF graph shows instances and refers back to where they are defined. For example, gray ovals point to the mainList for each instance. Predicates (arcs) point to the schema (Fig. 4.1) in which the respective relationship was defined. Literals like names were not defined anywhere, so they stand for themselves.

The name-relationship needs to be defined somewhere, and the problem description said you should assume they were defined in the ontoMed.schema ontology.

Inheritance is a relationship between classes (or types), but not instances. Each instance belongs to one type or class. The most specific class is referenced in the RDF graph. For "Weight Lifting," this is Physical Therapy as opposed to Treatment.

Problem 4.9.4: Multiple roles

Looking at the RDF graph above: Can an individual or instance have multiple roles (subject, predicate, object)? Substantiate your answer with an example.

Yes, this occurred to Peter Painful. He is the subject of the triple: Peter–receives–Weight Lifting, and at the same time, he is the object in the triple Nathalia–attends-to–Peter (this short notation for triples is not officially defined, as you know, but you should understand what it means).

Problem 4.9.5: xmlns

What does the statement (xmlns:medSchema="http://ontoMed.schema/") mean and how can that be used in subsequent statements?

A shortcut (medSchema) is defined. The given prefix of URIs is the "path" to that schema. After opening that schema as a namespace, subsequent statements can use it instead of the lengthy prefix (e.g., medSchema:prescribes instead of http://ontoMed.schema/prescribes).

This makes the XML code more readable and avoids inconsistencies or typos.

Problem 4.9.6: Attributes

What is the difference between an attribute "name" in a Protégé ontology versus that in an object-oriented class model? How are attributes assigned in both environments?

An attribute in an object-oriented class model is a part of a specific class symbol. If several class symbols have an attribute that is spelled identically, like name, those attributes are nevertheless different.

In ontologies, attributes are entities by themselves and are not dependent on a class. The attribute name, for example, can be defined once (e.g., consisting of first name and last name). Every class using this attribute actually refers to that same entity, not just different entities with the same identifier.

In class models, the attribute is written in the second part of a class symbol. An attribute in an ontology is created in a separate editor before it can be assigned to none, one, or many classes.

Problem 4.9.7: OWL-DL

OWL-DL is often used as the variant for ontologies applied in practice. Name the two other variants available and provide reasons that make OWL-DL preferable.

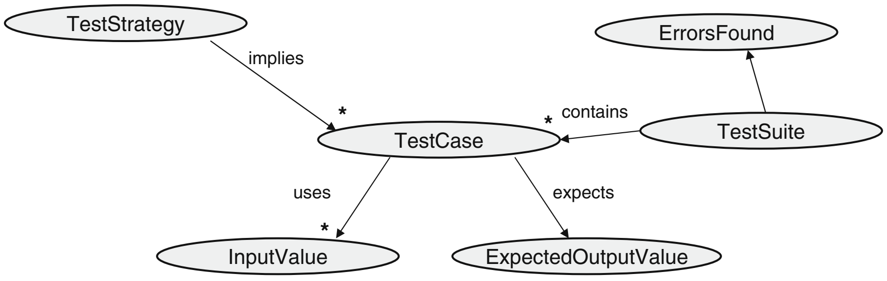
OWL-Full is powerful but not decidable. This makes it inadequate for formal operations.

OWL-Light is very lean but not powerful enough for many applications.

OWL-DL is decidable and powerful enough. This makes it the best candidate.

Problem 4.9.8: Test ontology

Construct a simple ontology in Protégé with six classes. Sketch the ontology first using the bubble notation of Fig. 4.1 or Fig. 4.3. The purpose is to collect knowledge on test strategies. A test strategy determines what test cases look like. A test case is characterized by the provided input values (parameter values) and by the expected outcome or reaction. A test strategy is considered successful if it helps to find many errors. The test cases used as a reference are combined in a test suite. After this sequence of test cases has been performed, the number of errors detected is recorded.



Problem 4.9.9: Knowledge acquisition

Who could provide what part of the knowledge you need to populate the above-mentioned knowledge base? Outline a plan for how you could get all required data into the knowledge base with the lowest possible effort.

Testers will be able to provide all data needed. They develop or use test strategies, derive test cases, and carry them out. When they analyze the results, they are able to count the number of errors detected.

Asking the testers looks straightforward. However, most of the data needed will probably be available anyway. In a mature software organization, test cases will be stored in a database, and so will test protocols. They contain errors and test cases performed (test suites, if you will).

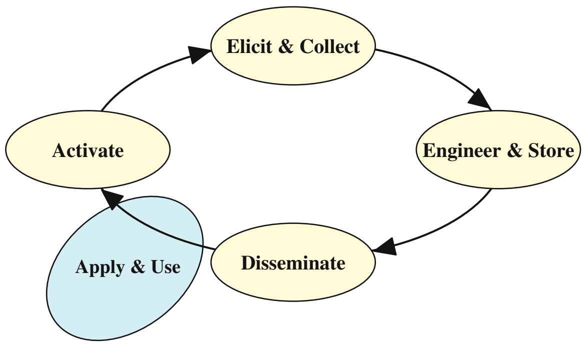
It should, therefore, be possible to get that data from the test databases using an automated routine. Remember to save everybody's time – except for the knowledge management professionals, who should invest a little of their time to save a lot of the software engineer's effort.

From a knowledge management perspective, knowledge acquisition will often mean asking people for their knowledge or data. However, as this problem shows, sometimes a little tool can free human actors from unnecessary work.

7.5 Chapter 5

Problem 5.6.1: Life cycle

Draw the experience life-cycle and briefly explain each activity.



The life cycle may start in any state/activity, depending on existing material at a particular point in time. Usually, doing something (Apply & Use oval) is the first activity. In terms of the experience life-cycle, activation comes first: After having done something relevant, the interesting insights, experiences, and knowledge need to be made conscious.

Activated experience, knowledge, and insights need to be assembled. Elicitation refers to the activity of actively asking and digging for those materials, whereas collection emphasizes the administrative or technical part of actually getting all the provided data together in an organized way.

Engineering and storage means adding value to knowledge and experience by relating it to other material and by modifying its style. While an experience is about an observation made in the past, engineered experiences are often turned into recommendations or best practices: guidelines on what to do in a similar future situation.

Dissemination goes beyond making something available: It refers to an activity of delivering material in a specific format to people at the time they need it. Under these circumstances, recommendations can be used when performing a project duty. At that time, new observations are made, and the circle may start again.

Problem 5.6.2: Experience engineering
Assume your team is using a new framework for software development. Programmers report problems they have had. If possible, they also report how they solved them in the end. What should experience engineering do with those reports, and what roles can patterns play?

Experience engineering is the activity performed after collecting material. Once material has been elicited or collected in another way (e.g., measurement), it needs to be related to other observations. Statements and conclusions in similar situations should be compared and combined into a more believable (or more differentiated) view. If all conclusions agree, this strengthens the conclusion. If there is a different view on a similar situation, differentiating factors and preconditions need to be investigated (either by reading the submitted material or by performing additional interviews). Contents are derived by interpretation and comparison.

After that, the result of this analysis needs to be "turned around" from stories of what happened into recommendations of what should be done (best practices). Patterns can help to format both experiences and recommendations. The situation is described as an "IF" part of a pattern, and the conclusion (or recommendation) is written as the "THEN" part. Patterns make reuse easier, because they explicitly factor out conditions and recommendations.

Problem 5.6.3: Experience three-tuple
Describe one of your own experiences in software engineering as an experience three-tuple. Make sure you do not neglect the emotion!

Observation: We changed an interface in a project. Only one type was "relaxed," so an operation could accept more instances than before. However, it turned out that two groups had used the initial interface definition in a way that later caused problems during integration. It took some time to understand how "relaxation" could cause problems.

Emotion: We were stunned and felt bad, because we knew very well about the importance of interfaces. All three groups had to make changes and corrections, which took some of their time. It was an embarrassing situation for those who had decided the interface change was "neglectable."

Conclusion: Never ever change an interface without informing all affected groups. Let them consider implications before you take the freedom to change what you have agreed upon. Only during experience engineering will the experience be turned into a practice recommendation.

Problem 5.6.4: Best practice

All projects in a business unit need to follow a new process. The first project using it reports some experiences. Why can experience engineering usually not be shortcut by asking the projects to provide best practices right away?

A single experience is usually not sufficient for generalizing from it. Like a theory, a conclusion needs to be confirmed to be believable. Only when it is a pure technical problem that was solved and only if there is little doubt about the proposed solution could this one conclusion be presented as a recommendation.

It is better to have one experience than none. However, there is little use in reversing it into a practice – and the label "best practice" is not appropriate if there is no comparison with any other practice at all.

Problem 5.6.5: Contradictory experiences

Two testers report experiences on a test tool. Tester A is happy, because the tool helped him to create far more test cases than usual. A larger number of errors were found early in the testing process, and late testing phases were far more relaxed. Tester B, however, was disappointed because the tool provided only "trivial test cases," as he puts it. Assume you are an experience engineer and need to make sense of those contradictory experiences. Give at least two potential explanations for the different experiences! If both testers A and B are equally competent, what could experience engineering ideally try to derive from their experiences?

Given both partners are competent, the different situation and context should be the source of the different experiences. An experience engineer will, therefore, analyze those differences carefully. Maybe one of the two people had different expectations or was in a different situation.

For example, code in example A may contain more errors, so the tool can be used more effectively for finding them. Tester B might already have worked with code B for a longer time and have better preexisting test cases. In that situation, the tool has more problems exceeding the present state. Present state, quality of the code, and many other properties can be differentiating factors.

The IF-part of a pattern will have to reflect that differentiation. Future projects will need to classify themselves, so that the best-matching patterns are found. They should contain recommendations best suited for the particular context and situation at hand.

Problem 5.6.6: Delicate experience

Why is experience an even more delicate matter than factual knowledge? Describe two different kinds of experience: (1) one highly critical kind of concrete experience that would be very helpful to reuse, but that will be very hard to elicit; (2) one kind *of knowledge that should be easy to get (and useful to reuse). As an experience engineer, what kind of experience would you focus on first?*

Experience is always subjective. Because of a certain perspective, details or aspects of the situation may have been missed during the observation. Conclusions may be faulty, and the emotional aspect is deeply subjective. A raw experience is difficult to validate, and the subjective elements are difficult to preserve during the experience life-cycle.

1. There are experiences that either remain unconscious or that are embarrassing for somebody. Those experiences are hard to elicit ("get out of somebody to"). For example, a project manager may find that she is weak in planning and did not know how to use the planning tools correctly. Although conscious, this experience will hardly be made explicit by that project manager – but others on the team may have made the same observation. If a person has been doing something for a very long time, there is little chance it will be considered an experience. The challenge for experience elicitation lies in helping this person: Seeing the value and the details in daily (implicit, tacit) knowledge is difficult. A highly "experienced" project manager may master his planning tools without giving it a thought. It takes some questioning to find out more about those procedures.
2. Easy-to-get experiences are about events and things that people were very aware of (trying something new, having a success or problem).

It is advantageous to take the easier material first. Use the experience management effort wisely! When the easy material takes only a short time for elicitation, the experience that is more difficult to elicit will still be rather fresh afterward. Because there is only limited time available for total elicitation, as many useful experiences as possible should be collected – not starting with one that will use up all the time assigned.

Problem 5.6.7: Argue well
How do you react if your boss asks you to start developing a knowledge-building strategy for your team of eight people – with the option of spreading it across the entire business unit if it really provides substantial benefits. Once you reach that level, he promises to provide additional resources and promote you to "knowledge expert." Sketch your argumentation!

Starting at the point where you are is a good idea. However, a company-wide knowledge management initiative can hardly grow from bottom-up alone. It has been reported many times how important management commitment is for knowledge management. Starting with a small team may lead to effective support for that team (e.g., resulting in a team glossary and a simple repository of experiences). Larger teams or entire business units call for additional and different mechanisms. The same strategy will not work (i.e., everyone who made an important observation fills in a form and puts it into the paper folder). Knowledge management practices do not scale up easily.

Therefore, you should take the chance to build something for your team but point out the above-mentioned argumentation. It is derived from experience in several companies. Ask for more commitment upfront or recommend setting more modest goals.

7.6 Chapter 6

Problem 6.7.1: Life cycle
A friend tells you they are using a newsgroup as experience base. Which of the typical tasks of an experience base can be performed by a newsgroup, and which cannot? Provide arguments for all examples.

- Activation: When a problem is discussed in a newsgroup, this may show some participants how relevant their experiences are.
- Collecting: Because communication in a newsgroup works through writing, nothing is lost and everything is collected.
- Engineering: A moderated newsgroup may contain elements that resemble engineering. However, most newsgroups offer the stored material more or less in the way it is typed in. There is no major reformatting or rephrasing.
- Dissemination: When the newsgroup offers mechanisms for distinguishing threads or sub-newsgroups, a reader has a better chance of finding relevant entries. However, there is usually no sophisticated mechanism for dissemination.

Problem 6.7.2: Risk of experience brokers
A situation like the one at Ericsson is risky: An experience broker may leave the company and disrupt the exchange of knowledge and experience. What could be done to mitigate that risk? That means: If a broker actually leaves, how will your suggestion improve the situation?

Let two or more brokers work together. That way, a lot of implicit knowledge can be exchanged without the need to document it. When one of the brokers leaves, the other one can take over.

Documenting everything is not such a good idea. A core concept is the fast and unbureaucratic response to a demand. Documentation would slow down and disrupt the reaction chain.

Problem 6.7.3: Risk mitigation
Name three important differences between a community of practice (CoP) and an expert network. What do they have in common?

- A CoP is a volunteer organization; an expert network is organized and supported by the organization.

- Entering and leaving a CoP is up to the participants. The members of an expert network are usually nominated by the company.
- A CoP is a network of people who work in the same domain. An expert network connects people whose expertise may complement each other.
- Participants benefit from a CoP by learning about other experiences. In the best case, some of those experiences are documented and made available to other projects. Members of an expert network are often supposed to provide advice or guidance to a new project. There is more direct support.

In both forms, there is a group of people defined by their knowledge. Connecting those people is considered a way to sustain implicit and tacit knowledge.

Problem 6.7.4: Compare
The LIDs technique is optimized for "cognitive aspects." Explain what that means and provide two concrete examples within LIDs.

Cognitive aspects refer to the human abilities (and disabilities) during the task of providing or reusing experience. Considering cognitive aspects should help to avoid unrealistic expectations and demands that cannot be met by most people.

For example, people cannot distinguish the crucial from the irrelevant aspects of an extended task. Without support (by an agenda or a table of contents), they may get lost.

People are not willing to spend long hours on capturing experience; they prefer to continue with the next assignment. Therefore, minimizing the LIDs session duration is a contribution to a cognitive aspect.

Problem 6.7.5: Cognitive aspects
Many knowledge management visions include the role of a knowledge manager. In the case of a software engineering knowledge base: What background should a knowledge manager have, and what existing role in a project (see Fig. 4.3) might be a good choice?

A knowledge manager should have an EKM background, as presented in this book. There should be a good overview of the techniques relevant for structuring knowledge and the ability to describe knowledge and experience in patterns and maybe ontologies.

In most cases, it is more important to have experience in setting up learning initiatives than to know all details of a formalism. The latter can be acquired faster than can good judgment about EKM options.

Quality people are often a good choice. They have a cross-cutting concern for quality. Many of them are organized in CoPs or other networks. And they should be organized in the quality hierarchy, which helps them to consider other project experiences.

Problem 6.7.6: Seeding

You have designed a knowledge and experience base about test strategies and their effectiveness. What could you seed this knowledge base with, and where do you get that knowledge from?

Seeding could start with a book or with single experiences that someone tells you in a meeting. With book material, make sure to keep advice concrete. With experiences, try to generalize them toward reusable information. The combination of both book (general) and specific aspects helps to make a good seed. Do not try to cover too many aspects, but rather to reach a certain depth.