Appendix A

Answers to Selected Exercises

Chapter 2: Systems

1. (a) The function of the grinder for its users is to grind coffee beans. (b) The environment of the system consists of its users, the electrical power supply, and the surrounding air. The interface to the user consists of the following interactions: accept an on or off transaction from the switch, switch light on or off, accept a volume of coffee beans, produce a volume of ground coffee. The interface to the electrical power supply is to draw electrical energy from the power plug, and the interface to the surrounding air is to produce waste heat. (c) The system is hybrid.

2. At the highest level, the interface consists of the transaction *sell grocery*. This is the way a branch organization would look at the store: the only relevant thing the branch organization wants to know about the store is that it sells grocery. At the next lower level, we observe interactions like *sell bacon* and *sell cheese*. This is the way the customer looks at the store: he or she wants to know which grocery can be bought at the store. At the next lower level, we can observe transactions like *enter shop*, *walk to shelf*, *get packet of cheese*, etc. This is the way an operations researcher may look at the store, who is interested in finding out the optimal placement of the shelves, of the goods on the shelves, etc.

6. The two diagrams represent different behaviors, because in diagram (a), the action c is nondeterministic: it may lead to one out of a set of two possible next states. From one of these states, action b can occur, and from the other d can occur. In diagram (b), the action c is deterministic: it leads to exactly one state, from which a choice between b and d can be made. Thus, the moment of choice is different in the diagrams.

Chapter 3: Product Development

1. (a) The client is the aircraft vendor. If the software is market-produced, the sponsor is the vendor of the software and the customer is the aircraft building company; otherwise the sponsor and customer is the aircraft building company. The user is the pilot of the aircraft.

   (b) The sponsor is the DBMS vendor. The client of the development process and customer of the DBMS is the buyer of the DBMS, the user is the person who must install and maintain the DBMS.

(c) The client and customer is the buyer of the software. The sponsor is the software vendor and the user is the library employee at the circulation desk.

(d) The client, sponsor and customer is the government in the form of the ministry of defense. The users are the officials who query the database.

2. (a) (1) Informal reasoning by managers. (2) Pencil and paper sketches. (3) Informal reasoning, performed by the designer about the likely consequences of certain design options. (4) Building a prototype. (5) Performing a trial production.

(b) (1) Informal evaluation after discussion by managers. (2) Informal evaluation after discussion by designers and managers. (3) Comparison with required properties by designers. (4) Performing a consumer experiment. (5) The evaluation criteria are not mentioned in the case study. Examples of criteria are repeatability, reliability and economic feasibility of the process.

3. Because business concerns are transformed into software requirements, this framework applies to the level of computer-based systems. *Elicitation* is the fact-gathering part of needs analysis, *specification* consists of analyzing the gathered facts and of synthesizing it into a requirements specification, and *validation* consists of simulation, evaluation and choice.

6. The finite number of observations of the disturbances were used as evidence for a model of Neptune and its orbit. This orbit contains infinitely many points. The step from the observations to this model is the inductive jump.

7. The correspondence with rational problem solving is as follows: **Problem analysis** corresponds to *suggestions* and *intellectualization*. That is, the subject becomes aware that there are other ways of doing things, that the current way need not be taken for granted, and makes the transition from unreflective and unquestioning doing things the current way to a disengagement, distancing from the current way of doing things, which opens the possibility of reflective thinking; and the subject then starts to analyze the current situation. **Solution generation**: *hypothesis*. **Estimation of effects**: *reasoning*. **Evaluation of effects**: *testing* the hypothesis by action. It is not clear from the description in the exercise whether this action is an experiment or the real thing. **Solution choice** is not listed explicitly in Dewey's process. The process can be viewed as a prescription for reflective action in which we alternate between reflection upon the current situation and performing an action. After the first action is performed, reflection upon the current situation can be viewed as part of the regulatory cycle (observe and evaluate the effects) as well as of a rational problem solving process that prepares for the next action.

8. *Scouting* and *entry* are pre-development tasks that precede the engineering cycle. *Diagnosis* corresponds to **needs analysis**. *Planning* corresponds to **synthesis of product specifications** and **Simulation of specified products**. Kolb and Frohman do not mention **evaluation of effects of simulations** or **choice of specification** explicitly, but these can be viewed as part if their *action* task. Their *evaluation* task is really part of the regulatory cycle. *Termination* is a post-development task that follows the engineering cycle as well as any applications of the regulatory cycle.

Chapter 4: Requirements Specifications

2. (a) In the first iteration, the documented objective was to preserve TANA market share. The alternatives considered were to search new markets for existing products, to develop new products for the same market, and to adjust the marketing mix. The likely effects of these alternatives were simulated and evaluated with respect to the objective by management. The discussion is likely to have been recorded in the form of minutes of a management meeting.

(b) The alternatives considered are listed in figure 3.7. The simulations were their own documentation, because they are paper and pencil sketches. Their evaluations are probably recorded as minutes taken at a meeting in which management evaluated the sketches against the marketing objectives.

Chapter 5: ISAC Change Analysis and Activity Study

2. Figure A.1 gives a solution. The activities are systems with local memory, which resemble the objects of object-oriented modeling. The arrows indicate messages sent by one object to another. This model is simpler than the one in figure 5.8, because it contains less interaction, and the interaction that it contains is essential for the functioning of the system. It has the disadvantage that data are distributed over different activities, whereas in fact they are located in one place (a paper or automated database). The diagram thus has less resemblance than figure 5.8 to the way that work is currently done in the circulation system.

4. Alternative 2 of the library case study (section 5.2.8, the store room alternative), has the same activity model as the current situation.

Chapter 6: Information Strategy Planning

4. See figure A.2 for an example. This is not the only solution; in fact, the tree may differ for different travel agencies.

5. See figure A.3 for one possible solution. A segment is characterized by a point of departure and a point of arrival. A scheduled segment in addition has particular departure and arrival dates and times, which are not represented in the diagram because they are attributes and not entity types. It is not clear from the exercise whether the segments of one flight must be consecutive in space and/or time. A reservation can profit from the price arrangements by reserving a number of segments such that a Saturday night falls in the time between departure from the starting point and departure from the destination. A complex reservation consists of a reservation for several segments. Each complex reservation is a reservation for one person. A complex reservation has a price that depends upon the departure and arrival dates of some of its segments. The price is not shown, because it is an attribute of complex reservation. Return flights have not been modeled explicitly. They can be defined as a view on $FLIGHT$ instances, defined by the predicate that the sequence of segments of the flight starts and ends in the same airport.

6. Example subject areas and entity types that belong to those areas are:

   - $CUSTOMERS$: $PERSON$
   - $FLIGHTS$: $AIRPORT$, $FLIGHT$, $SEGMENT$, $SCHEDULED\_SEGMENT$
   - $SALES$: $COMPLEX\_RESERVATION$

   The $FLIGHTS$ subject area corresponds to the products subject area of figure 6.14. Note that there is no reason to stick to the list of example subject areas of figure 6.14. Any partitioning in subject areas that the client agrees with, suffices.

7. See figure A.4. The matrix covers only two of the business areas of the travel agency, Sales and part of the Acquisition business area.

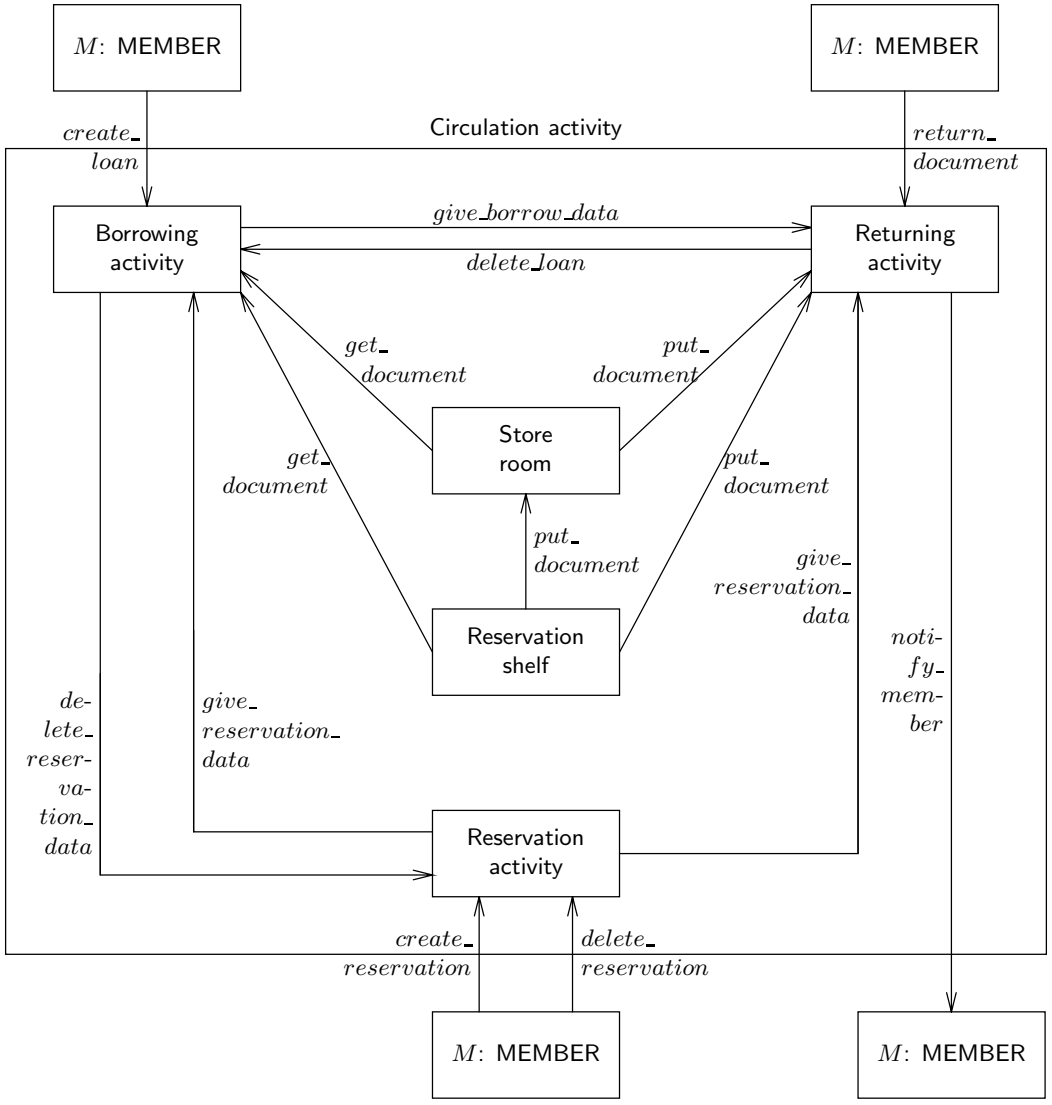


Figure A.1: An activity model of the circulation system that encapsulates memory in the activities.

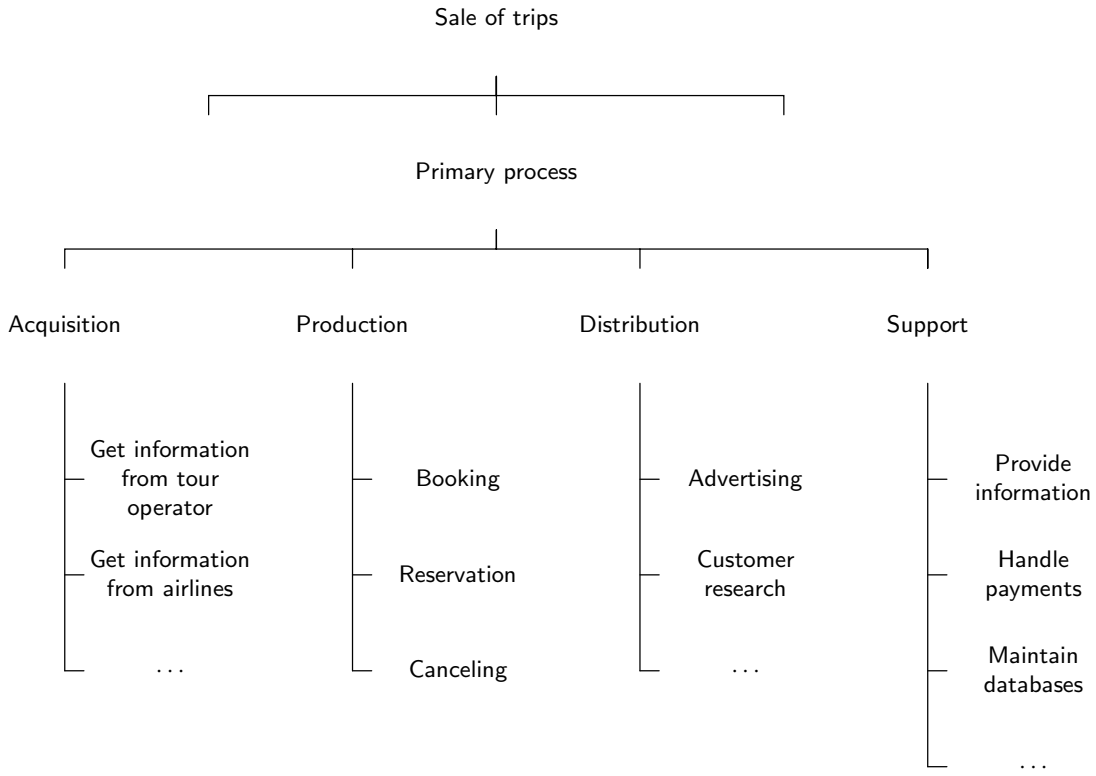


Figure A.2: A function decomposition tree of the primary process of a travel agency.

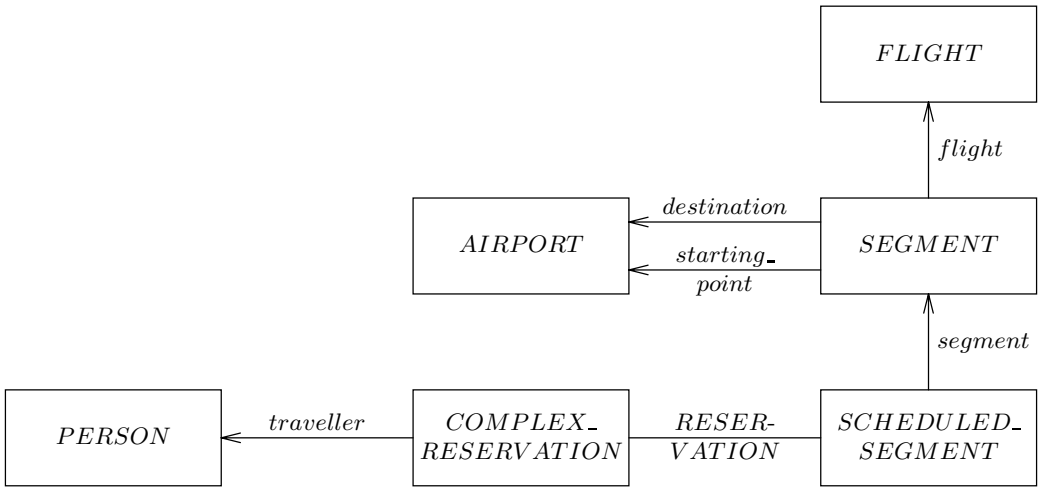


Figure A.3: An ER diagram of the UoD of a travel agency.

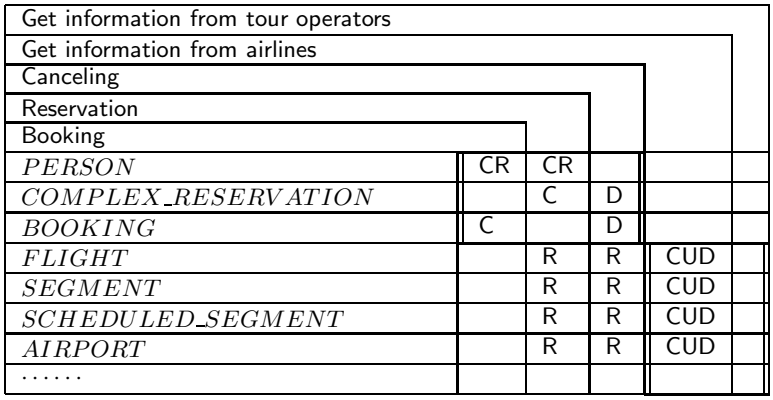


Figure A.4: Fragment of a function/entity matrix for a travel agency.

Chapter 7: The Entity-Relationship Approach I: Models

2. In the UoD, each existing airline has at least one existing flight. There exist cities without airports, addresses without people having that address, and flights and people without reservations. The only visible constraint is thus ≥ 1 at the $FLIGHT$ side of *airline*.

   In the system, each existing airline has at least one existing flight. We want to represent cities without airports, but we do not want to represent addresses without people. The other cardinalities are the same as in the UoD. Thus, we have ≥ 1 at the $FLIGHT$ side of *airline* and ≥ 1 at the $PERSON$ side of *address*.

4. The second one cannot be represented, because a cardinality constraint at the root of the arrow *supplier* says how many links can occur in which a single supplier s can occur. It does not and cannot say anything about the number of links in which s and a part p occur.

5. If attribute a is not applicable to some instances of entity type E_1, remove a from the definition of E_1 and define a specialization E_2 of E_1 for which this attribute is defined.

6. Replace a partial function by a relationship in diamond notation.

7. Define a relation $TRANSPORTED\_DELIVERY$ with components $COMPANY$ (playing role *transport_company*) and $DELIVERY$, which itself is a relation with components $COMPANY$ (playing role *supplier*), $PART$ and $PROJECT$.

8. Turn *LOAN* into an entity type. It then has its own identity and there can be arbitrarily many *LOAN* instances with the same member and document. The cardinality constraints that one member borrows at most 20 documents, and one document is borrowed by at most one member, are not expressible anymore in the diagram.

9. The referential integrity constraint allows the referring key to be *null*. A component of a relationship must never be *null*.

10. If a many-many relationship is transformed into one relational schema, it is not in the fourth normal form. A relationship not in fourth normal form contains two independently varying multi-valued facts. See also Kent [175].

11. - Less then 100 students follow course CS101 every year. This can be a descriptive regularity in the UoD. Treating this as a constraint on the system, we get a system that cannot be used in a situation where more than 100 students enroll for CS101. We

should only turn this sentence into a system constraint if we can be absolutely sure that the UoD regularity expressed by the sentence will never be falsified.

- Course CS100 is a prerequisite for course CS101. This may or may not be an official rule. If it is not an official rule, it is a descriptive sentence that expresses an observation made of the UoD. If we were to turn this into a system constraint, we would turn this observation into a UoD regularity that will never be falsified; which is a risky decision. If the rule is a UoD constraint and we want to enforce this by means of the system, we should turn it into a system constraint. Exceptions cannot be recorded. If the rule changes, we must then change the system specification.
- Students always follow CS100 before they follow CS101. This may be a consequence of a rule or a consequence of a habit of the students; which habit may follow from the fact that CS100 really is a prerequisite of CS101, even if the rules don't say so. The same considerations as above apply.

Chapter 8: The Entity-Relationship Approach II: Methods

1. The first one can be represented by adding the cardinality ≥ 1 at the root of the arrow $SALE \rightarrow CLIENT$. The other constraint says that the multivalued function $CLIENT \times SHOP \rightarrow\!\!\rightarrow PRODUCT$ defined by $SALE$ assigns at most two products to any particular $\langle c, s \rangle$ and this cannot be represented in the diagram.

4.
   - This is a car: Instance-type relationship.
   - A Ford is a car: is_a relationship.
   - A Ford is a type of car: Instance-type relationship, since "Ford" is now treated as the name of an individual.
   - A teacher is a member of staff: is_a relationship.

5. Add relationships $RECEIVE\_MARK\_FOR\_TEST$ and $FINISH\_TEST$ that represent the events of obtaining a result. The relationships $PRACTICAL\_REG$ and $TEST\_REG$ represent the events of registering for a practical and a test, respectively, and allocation of a *result* attribute to these events is a type error.

7. The key of the relational schema corresponding to a relationship can now have components that are themselves compound, and referential keys may now also be compound.

Chapter 9: Structured Analysis I: Models

3. (a) Material stores have only two operations: remove and add. Removal is a destructive read and corresponds to the effect of a read and delete operation on a data store. Addition corresponds to a creation operation on a data store. Data stores have non-destructive reads and additionally have an update operation, which is absent from material stores. Material flows move material items just like data flows move data items. However, if masses are manipulated (like water), then the flow is continuous and we need an additional notation to distinguish these from discrete flows. Material manipulations are physical processes that must be specified by means of notations from physics, chemistry, biology, etc. They can be discrete or continuous.

   (b) There is no difference with material manipulation. The dashed line is therefore superfluous and can be replaced by whatever notation is used for material flows.

Event	Response	Transaction
Member requests title reservation	Reserve title	Reserve title
Member requests cancellation of title reservation	Cancel title reservation	Cancel title reservation
Member requests to borrow a document	Lend document	Borrow document
Member requests to extend a document loan	Extend loan	Extend loan
Member returns a document	Accept document return	Return document
Document loan is overdue	Send reminder	Send reminder
Member loses document	Register document loss	Lose document

Figure A.5: Event list for the ciculation administration.

Chapter 10: Structured Analysis II: Methods

1. When a document borrow request is received, the administration must check whether it has been reserved (read access) and if so, whether the borrower is the reserver; if the borrower is not the reserver, the document cannot be borrowed, otherwise a reservation record must be deleted (write access). This is one data store access in which records may be read and updated at the same time.

3. See figure A.5.

4. 1. Feasibility study is a behavior specification process at the level of computer-based systems.

   2.(a)–(c) Structured analysis is a behavior specification process at the level of computer-based systems that makes the result of feasibility study more explicit.

   2.(d)–(i) This is a decomposition process that follows the engineering cycle. A computer-based system is decomposed into manual tasks and software systems. The observable behavior of the software systems is specified.

   3. Structured design is a decomposition process that decomposes software systems into modules.

   4. Structured programming is a decomposition process that decomposes software modules into executable parts.

5. 1. This is a behavior specification process at the level of computer-based systems. Needs analysis is already assumed to have taken place, as this task begins with understanding system objectives that are already previously identified.

   2. Building a processor environment model is a decomposition task as well as a behavior specification task for the units into which the system is decomposed.

   3. Specifying the human-computer interface is a behavior specification task at the level of software systems.

   4. Specifying a software environment model is a decomposition and behavior specification task, in which each unit is decomposed into software subsystems called tasks and the behavior of these tasks is specified.

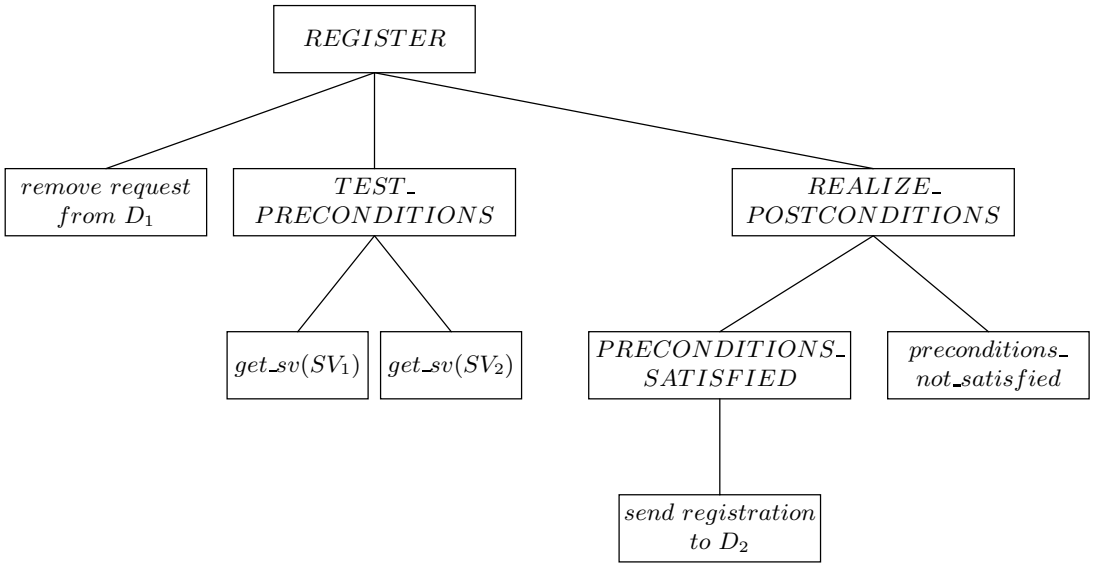


Figure A.6: Outline of a PSD for the $REGISTER$ function.

5. Structured design is a decomposition of tasks into units of code.

Modern structured development decomposes software systems into tasks, and tasks into modules, where classical structured development decomposes software systems into software modules immediately.

Chapter 11: Jackson System Development I: Models

1. Input: A registration request of student s for test t.
   Precondition: s and t exist and the *register* action can occur in the life of t.
   Postcondition: The registration request has been removed from D_1 and a registration record has been sent to D_3.

   Precondition: s or t do not exist or the *register* action can occur in the life of t.
   Postcondition: The registration request has been removed from D_1 and a refusal has been sent to D_2.

   Figure A.6 gives an outline of the $REGISTER$ PSD.

2. The $REPORT$ subtree of the $LATE$ PSD is a short-running version of $LATE$. The connection cardinality must be changed so that the ≥ 0 cardinality is on the $LATE$ side.

   To turn $ACCT\_HISTORY$ into a long-running function, make the short-running function one iteration in a process that is triggered by the arrival of a *historical_query*. The connection cardinality changes so that one $ACCT\_HISTORY$ instance is connected to many $ACCOUNT$ instances.

3. This requires addition of a JSD entity $REGISTRATION$. In ER terms, this is a relationship between the ER entity type $STUDENT$ and the ER entity type $TEST$. The *register* action is the creation action for this relationship. The $REGISTER$ function of figure 11.21 is a function process that performs the input transactions in which this creation action is executed. The PSD of $REGISTRATION$ is trivial and consists of a sequence *register*;

perform; *mark*. We should add the *perform* action to the *TEST* life cycle as well. The *REGISTRATION* life cycle then shares all its actions with the *TEST* life cycle but enforces an ordering per test–student pair.

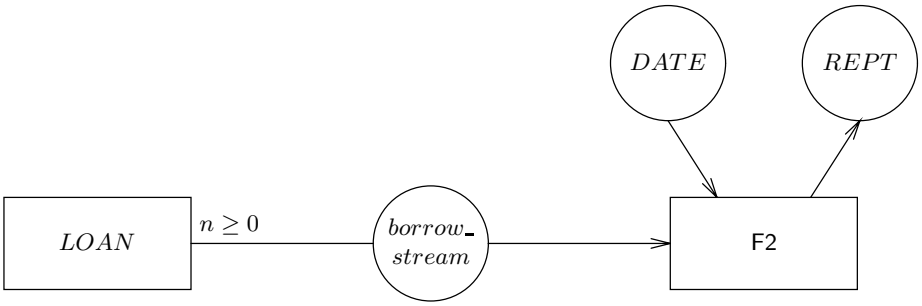
4. What is required is that allocation must be a synchronous communication between *ROOM* and *ALLOCATE* instances. Turn the data stream D_2 into a controlled data stream that locks a *ROOM* instance, observes the state of the instance, and then updates the *ROOM* instance.

Chapter 12: Jackson System Development II: Methods

1. This requires a periodic function process that scans all outstanding reservation processes and cancels those that are past their final date. The connection with the reservation processes must be by controlled data stream, because the deletion is conditional upon the state of the deleted processes.
2. The structure of *D_RESERVATION* is the same as that for *T_RESERVATION*. There are now two *res_borrow* actions, *t_res_borrow* and *d_res_borrow*, and the normal *borrow* action has an extended precondition that tests that none of these other cases is true.
3. The *FINE* life cycle now starts with a *lose* action and continues with an iteration over *lose* and *pay* actions. A fine is only created upon the first document loss.
4. This is a historical query that is periodically triggered by a temporal event. It is therefore a long-running function. Because it is a historical query, we cannot scan model processes on their current state using a state vector connection, but we must keep a log of the relevant action occurrences in a data stream. In addition, the function process will take input from the clock, to be able to notice the end of a week. The system network and a possible process structure for the periodic borrowing report function are shown in figure A.7. This process assumes that there is always at least one borrowing in a week. If this assumption cannot be made, a provision for a null report must be added, similar to the *null* action of F1. The *LOAN* process must be extended with the action *add borrow to borrow_stream* immediately after the *borrow*(and *borrow_res* actions (figure 12.7).

Chapter 13: A Framework for Requirements Engineering I: Models

1. (a) There is not enough information to put any other constraint in the model than $n \geq 0$.
   (b) A possible subject area *PRODUCT_OBJECTIVE* is the tree whose root is *GOAL*. The rest is then part of the subject area *PRODUCT_ENVIRONMENT*.
2. Figure A.8 gives a possible trace of a model containing one student s, one course c and one test t. Each column shows a history of an object, with the earliest action occurrences at the top. Synchronization by common actions is indicated by horizontal lines. Actions in the history of different objects not connected by a horizontal line may occur in any order (compatible with the synchronization points). The student receives a mark twice, without doing a test in between. This possibility remains when there are many students, many courses and many tests.
4. (a) Figure A.9 shows part of a specification of a simulation of a data stream. The connected processes are represented by external entities. The data store contains records labeled by a sequence number, that indicates the sequence in which they were put in. It



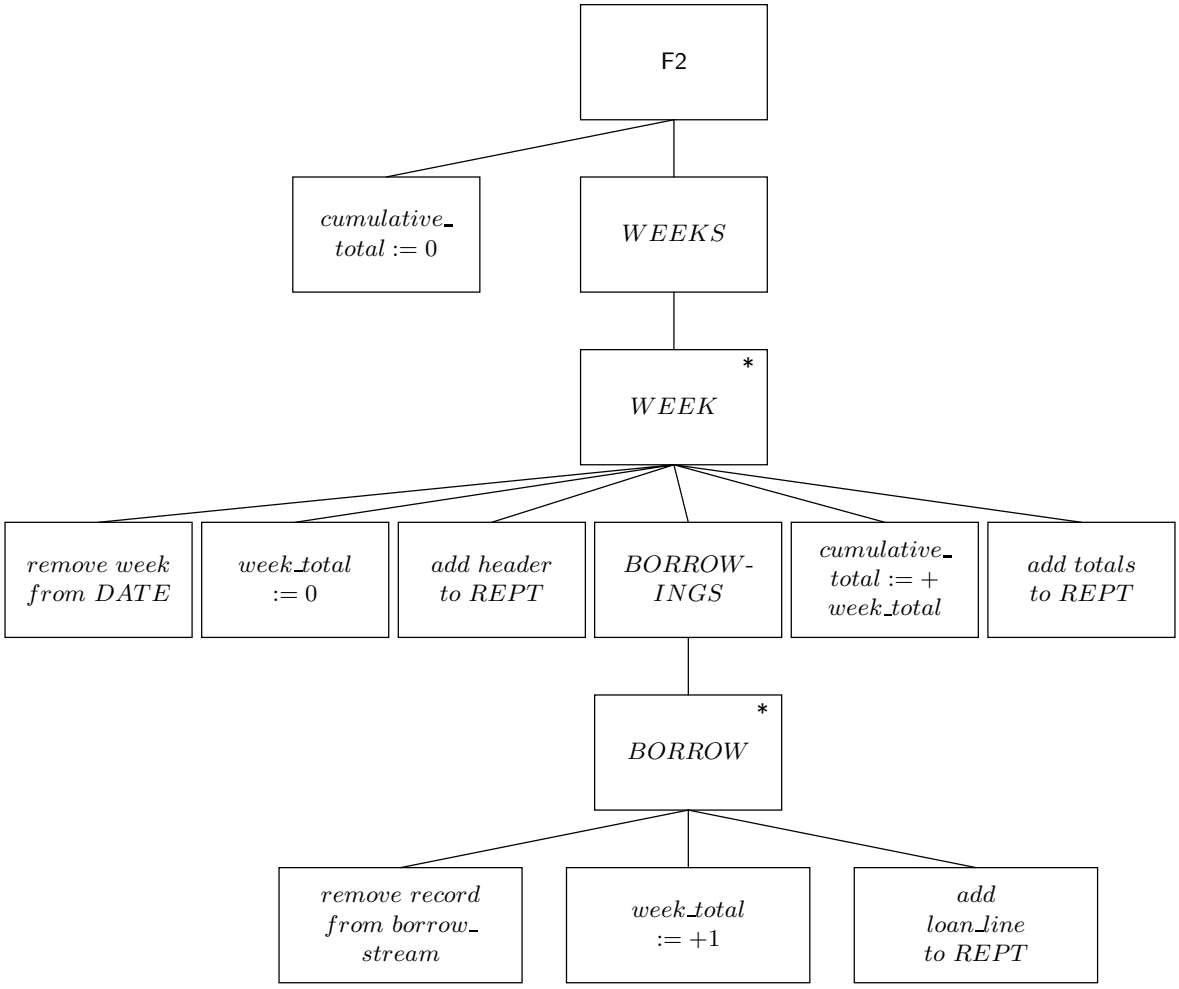


Figure A.7: System network and process structure for the periodic borrowing report.

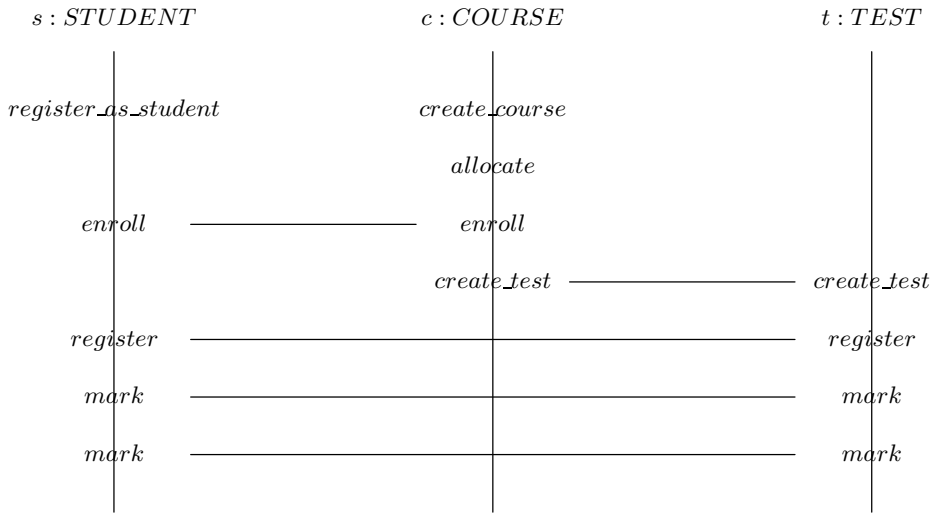


Figure A.8: A possible history of the student administration.

starts with the next sequence number to be given out and then contains the set of records written to it but not yet removed. This model cannot simulate the case where records are not stored but passed immediately, because addition and removal are then one atomic action. In the DF simulation, they are two atomic activities (functional primitives) performed in sequence. Moreover, The DF simulation cannot adequately represent the cardinality of the data stream connection.

(b) Figure A.10 shows a simulation of a controlled data stream. Records are passed immediately, depending upon the current state of the receiver. The conditional update is a functional primitive and is therefore atomic. (Just like the controlled data stream connection in JSD, it may have to wait till the observed process sends it its state vector.) However, it cannot represent the cardinality of the connection.

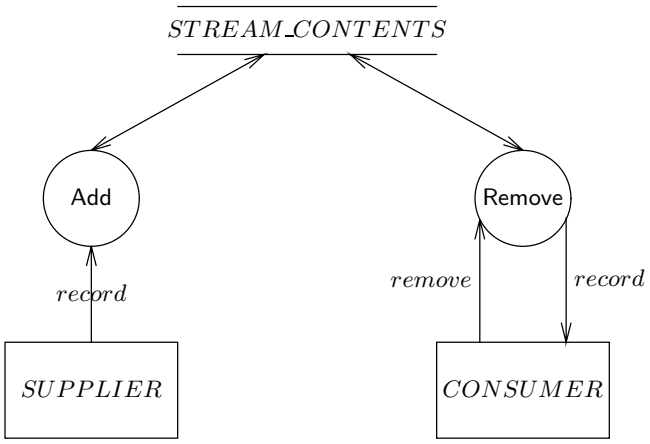
5. (a) Each JSD entity type corresponds to a JSD entity type or to a relationship. To complete it, we should check whether there are relevant actions in the remaining ER entity types and relationships.

(b) Each functional primitive corresponds to a JSD action in the UoD model. There are some minor differences in naming, due to the difference in perspectives of JSD and DF modeling, that can be easily resolved.

(c) Figure A.11 contains the DF diagram and the minispecs. The minispecs use pseudocode that is sufficiently clear for a programmer to code the functions.

Chapter 14: A Framework for Requirements Engineering II: Methods

1. As explained in subsection 12.2.1, the action allocation table omits all R's because these represent the testing of preconditions. As indicated by the action allocation heuristics in

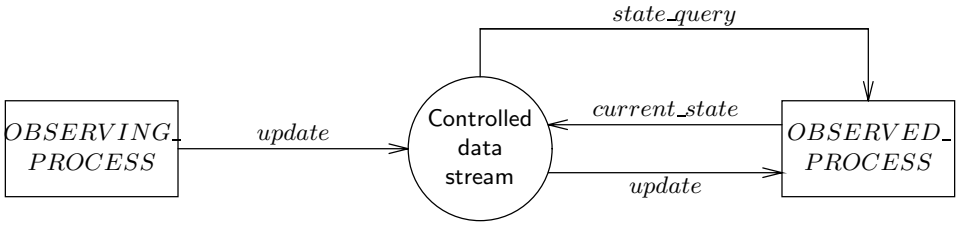


$STREAM\_CONTENTS = next\_sequence\_nr + 0\{record\}next\_sequence\_nr - 1$
$next\_sequence\_nr = NATURAL$
$record = sequence\_nr + \cdots$
$trigger =$ "Remove record"

Add
Read $record$ from $SUPPLIER$,
read value of $next\_sequence\_nr$ from $STREAM\_CONTENTS$,
write value of $next\_sequence\_nr + record$ to $STREAM\_CONTENTS$
increase $next\_sequence\_nr$ by 1 and
write it to $STREAM\_CONTENTS$.

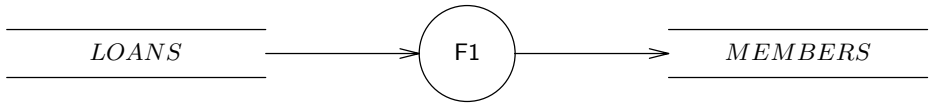
Remove
Read $trigger$,
Read $record$ with highest $sequence\_nr$ from $STREAM\_CONTENTS$,
Remove this $record$,
Write it to $CONSUMER$.

Figure A.9: A DF simulation of a data stream.



Conditional update:
Read $update$ from $OBSERVING\_PROCESS$,
write $state\_query$ to $OBSERVED\_PROCESS$,
read $current\_state$ from $OBSERVED\_PROCESS$,
if $current\_state$ satisfies the update condition of the controlled data stream
then write $update$ to $OBSERVED\_PROCESS$.

Figure A.10: A DF simulation of a controlled data stream.



F1:
Each day at 9:00 do:
search $LOANS$ for records with $date\_last\_reminded \leq today - 6\ weeks$,
for each of these,
read $pass\_nr$ of member from the record,
write $suspend$ to this member in $MEMBERS$.

Figure A.11: DF diagrams and minispecs of F1.

figure 12.3, testing the state vector of an entity type is not sufficient reason for allocating the action to that entity.

Chapter 15: Development Strategies

2. (a) To generalize to arbitrary client-driven development, replace the programming task by the construction task.

   (b) To change it into a market-driven strategy, the acceptance test is performed by the marketing department. Often, this is followed by β testing at selected customer sites, after which a release follows.

3. (a) Specification of required properties, performed during the initial stages of development.

   (b) Planning, performed during the initial stage of development.

   (c) Producing a user's manual is part of product development when the product to be delivered consists not only of executable software but is an installed system. It is performed during the implementation stages, starting from architectural design.

   (d) This is part of architectural design, as a preparation for the selection of algorithms in the next stage.

   (e) This is part of the programming task.

   (f) This is a task belonging to organizational implementation of the developed product.

   (g) This is part of the coding task, performed during the initial stage of development.

   (h) This is part of architecture design and performed during the corresponding stage.

5. Just like evolutionary development, the craftsman changes his design (1) in small steps and (2) in response to a mismatch between the product and user requirements that is experienced in actual use. However, (3) in evolutionary development, the product is then changed to reduce this mismatch. Wagons are not so easily changed as software is, and in general the learning experience led to an incrementally changed design for the *next* wagon to be built. In this respect, wagon-making is more truly evolutionary than evolutionary software development. Evolution is the property that *populations* adapt themselves to their ecological niche. As a metaphor, this is more accurate for traditional wagon-making than it is for software development. (4) A second difference is that there is no global product idea, part of this is then worked out. At any point in time, there are already wagons in use. (5) A third difference is that no global design of the wagon is ever made. This means that no global change to the design can be planned, nor can the likely effects of such a change be estimated.

Changes must be local and only incrementally different from designs that are known to work from experience.

The difference with experimental development is (6) that the customer requires a robust product. The livelihood of the customer depends in part on the quality of the product, so the customer doesn't want any surprises. This is another explanation for the incremental nature of design changes.

6. (a) ETHICS only contains product development tasks.

   (b) The entire ETHICS method deals with the social system level. The two questions that it tries to answer are: 1. How can the efficiency and job satisfaction needs of the organization unit be met? and 2. How do we decompose the organization unit into social and technical subsystems so that these needs are met?

   (c) ETHICS contains an exact match with the engineering cycle: tasks 1 to 9 correspond to **needs analysis**. Tasks 10 and 11 correspond to **synthesis**, **simulation** and **evaluation**. Task 12 corresponds to **choice**. This ends the correspondence with the engineering cycle. In terms of our development framework, task 13 is a decomposition task that brings us to a lower level of aggregation.

   Task 14 implements the design and task 15 evaluates the implementation. Thus, tasks 13, 14 and 15 correspond to the regulatory cycle.

7. (a) All tasks that say which documents must be written and whom to consult at which point in time are process management tasks. In particular, the *assemble report* tasks are process management tasks.

   (b) **Social system level**: feasibility study and requirements analysis. **Software system level**: requirements specification. **Decomposition of software system**: investigate technical system options (410, 420). **Software subsystem level**: specify logical design. **Software subsystem decomposition**: physical design.

   (c) Feasibility study and requirements analysis are two iterations through the engineering cycle at the social system level. After the preliminary cycle (feasibility study), requirements analysis performs a thorough needs analysis (investigate current environment), generates solutions, simulates and evaluates these (cost/benefit analysis and impact analysis in task 210) and chooses one.

   Requirements specification does not match to the engineering cycle. It can be viewed as a conceptual modeling task, in which a vaguely specified solution is modeled more explicitly. This is a descriptive activity; the normative questions have been dealt with in requirements analysis (task 2).

   Task 4 (investigate technical system options) is a decomposition that follows the rational problem solving cycle.

   Task 5 (specify logical design) is a conceptual modeling task at the level of software subsystems.

Chapter 16: Selecting a Development Strategy

5. (a) *Spiral method.* The growth envelope is limited, understanding of the requirements is high and robustness of the system must be very high. Depending upon available product alternatives, transformational development or waterfall development are possible options.

   Euromethod. Although this is an embedded system and not an information system, it is nevertheless illuminating to apply the Euromethod heuristics. The initial state

is that there exist requirements for the current generation of control systems. These are well-understood and the project aims at reimplementing these systems using new technology. According to figure 16.6, this should be performed in several adaptations. Due to the high safety requirements and the strategic importance of of the system, risk must be reduced to a minimum. It is decided to perform two adaptations, one in which a global system design is produced and a second one in which this design is implemented.

In both adaptations, the complexity of the manipulated information is low; however, the target domain factor more suitable to this domain is the complexity of the control structure of the system, and this is complex. Furthermore, the strategic importance of the system is high, which increases the uncertainty about the system. Complexity and uncertainty are further increased because the complexity of the target technology is high and the technology is novel. Uncertainty arises in the project domain as well, because the *development* technology is complex and relatively unknown: the company's engineers have little experience with computer technology. All of this leads to the selection of an incremental construction strategy: Requirements are known but will be implemented incrementally for each of the two adaptation processes. Note that the second adaptation process (from global design to implementation) can start as soon as the first process delivered its first increment of the global design and that from that point onwards, there will be two parallel adaptation processes until the global design is finished.

In this example, Euromethod leads to a more conservative advise than the spiral method. This is mainly due to the larger number of factors considered by Euromethod, which leads to a sharper focus on the project risks and thus to more risk-avoidance.

(b) *Spiral method.* The growth envelope is large and the understanding of requirements is low. Robustness of the system must be high and there is, at the higher levels of network technology, no technology of reusable components. (At the lower levels, standard network software is available on the market.) Architecture understanding is low. The spiral method heuristics lead to risk reduction followed by waterfall as one possible strategy, and the full spiral method as another. Due to the high robustness desired of the system, risk reduction followed by waterfall is chosen.

Euromethod The difference between initial and final state leads to a decision to perform more than one adaptation. One possible sequence would be the production of an information system change study first, the production of a global design next, the production of a tested system next, and finally the production of an installed system. For any of these adaptations, the information system complexity factors all score on the high side; the information system uncertainty factors all score on the uncertain side. Due to the many different systems in operation, the complexity of the computer system is high. The uncertainty about the computer system is medium to low, for all computer technology to be used in the project is commercially available. The complexity of the project task is high and the uncertainty of the project is medium to high. The number of interfaces is potentially high. Project uncertainty is also increased by the potential dependency on subcontractors and on other IS adaptations. Given the complexity and uncertainty about the target domain as well as about the project, the project manager would do well to keep the complexity and novelty of the development technology low. In the face of all of this uncertainty, the Euromethod heuristics recommend evolutionary construction.

The difference with the spiral method recommendation can be explained by the fact that risk reduction, recommended by the spiral method, is *always* recommended by Euromethod (we did not treat risk reduction in chapter 16). Furthermore, the sequential

nature of the waterfall process is also present in the Euromethod advice to perform several adaptations in sequence. The Euromethod advice can be viewed as a refinement of the spiral method advice by recommending evolutionary construction.

Appendix B

Cases

B.1 The Teaching Administration

A teaching administration maintains data about courses, practicals, tests, examinations and students. Each course and each practical is given regularly and there are one or more tests for each course. An examination consists of a number of tests. The examination is done by doing these tests. Examinations are done once every month and consist of a ceremony in which students receive proof of having passed the examination. When a student hands over proof of having done these tests with sufficient result, he or she has the right to participate in the ceremony.

To be able to do a test for a course, a student must register for the test. Each test has two or more supervisors, who check that the test participants are registered for the test. Students must register for a test at the teaching administration. One day before the test is conducted, the teaching administration produces a list of test participants, and this is given to the supervisors. When students come to a test unregistered, they are sent by the test supervisor to the teaching administration to register. The student receives a late registration slip from the administration, which he or she can show to the supervisor as proof of registration.

There are members of staff, called counselors, whose task it is to monitor student progress and help students when there are problems with their progress. Every six months, the results of all students are aggregated in a report and discussed by student counselors. Students whose progress is too slow, are called to their counselor for advice. In addition, each student receives an advise about study continuation after his or her first year of study. Every year, each student receives a report about his or her results so far.

B.2 The University Library

The functions of the Free University Library are (1) to acquire documents containing information that is of use for scientific research and education, (2) to catalog these documents, (3) make them available, (4) preserve them, and (5) to act as custodian of the documents it acquired. The collection is made available not only for the Free University but for any scientific research or education at all. Other universities and colleges have the right to use

the library, and as a matter of fact do so. The only prerequisite for getting registered as a library user is showing a valid proof of identity, so private individuals can use the library as well. However, most users are students or staff at the Free University.

The library is divided into departments that more or less reflect the structure of the university in departments. Thus, there is the Biology library for the faculty of Biology, the Mathematics and Computer Science library for the Faculty of Mathematics and Computer Science, etc. Department libraries are located close to the Faculty they serve. Department libraries are grouped together into Scientific Area libraries. For example, there are α, β and γ areas. Libraries in one scientific area have a common administration.

A user is someone who has a reader's pass. Any student or employee can acquire a pass, as well as citizens who are not otherwise related to the university. A group of employees can also acquire a pass, called a "group pass". There should be one person accountable for the actions of this group, but any member of the group can borrow a book.

The library acquires a wide diversity of items, such as books, journals, series (e.g. the Springer Lecture Notes in Computer Science), Proceedings, internal reports from their own or from other universities, unpublished reports from research laboratories, Ph.D. theses, maps, microfiches, videotapes, old manuscripts, newspapers, microfilms, etc. Some of these are acquired for students (often several copies), most of these are for research purposes. Books themselves can come into a great variety of forms, such as multivolume works, multi-edition works, and even works that appear as one volume in one edition and as several volumes in the next — after which only volume 1 goes through successive editions.

Most books can be lent to users, but some can only be read in special rooms. Similarly, old volumes of journals, when bound, can be borrowed, but loose issues cannot be borrowed. Borrowable items receive a unique code so that they can be traced to a borrower.

A user can borrow a book for three weeks. Researchers can in addition borrow it for three months. At the end of the allowed lending period, a user should return the book or else renew the borrowing. Renewal can only be done when there is no reservation for the book. If a user does not return of book or does not renew the lending period, action is only taken after 1 extra week, by sending him or her (or them) a reminder. So a user is reminded of his obligation to return the book 4 weeks after it was borrowed. If it is not yet returned or renewed, a second reminder is sent after 7 weeks. After the second reminder, the user has still one week to respond. If one week after the second reminder there is no message from the user, he or she must pay a fine of Dfl 70 and is not allowed to borrow any more books until the book is returned and the fine is paid.

Any user can reserve books that are currently borrowed by someone else. He or she will receive a message when the book is available and the library will hold the book for ten days so that this user can borrow the book. If the book is not fetched after ten days, the book is returned to the shelf. There can be at most one reserver for a book.

If a user loses a book, he or she has to report this to the administration, who will issue an invoice for the price of the book. If a user loses a pass, the pass is registered as lost and the library will issue a new pass at no cost. If the lost pass is found, the user has to return it to the library. Journal issues can be lost as well. Since issues are not lent to users, this cannot be attributed to any particular reader.

For some years now, the library experiences problems that cause increasing hindrance to library staff as well as users and that hinder the library in the realization of its primary function, making scientific documents available to its users. An unknown number of books and journal issues is lost or stolen, and often it is not known which of the two is the case.

Sometimes, a book registered as present cannot be found on its shelf and there is no record of it being borrowed to anyone. On the other hand, a book registered as borrowed or even as stolen may be found on a shelf. There are no reliable statistics of the use of documents and of their availability, such as the ratio between reservations and borrowings. Availability of documents is further decreased because some users, especially University staff, lend books for months or even years without bothering to return them.

These problems have budgetary consequences, for lost or stolen documents must be replaced and the costs of this are added to the normal costs of paying for journal subscriptions and the acquisition of books. In the coming years, the library budget for the university is not likely to increase, to put it mildly. In view of cuts in university funding by the government, the budget will probably decrease in the next few years. At the same time, scientific publishers start new journals almost every month, and tend to double the subscription rate every few years. Most subscriptions are in US dollars, and due to fluctuations in the dollar rate, this price increase may pass unnoticed in some years and hit extra hard in other years.

Some faculties chronically overspend their budget by simply refusing to terminate subscriptions. Especially faculties of "old" sciences such as Chemistry, Physics and Mathematics have some very expensive reference journals and in addition a wide assortment of subscriptions that cover some specialities within their science very well. However, those faculties argue that they have a minimal subscription portfolio already, and that termination of more subscriptions would endanger the quality of their scientific research, which is of a high level. All faculties, young and old, argue that they only have subscriptions to a fraction of the available journals, and that it would be irresponsible to terminate even one of them. For some of the older faculties, some of these subscriptions were started in the nineteenth century and the library and faculties all agree that it would be a shame to terminate such a subscription. However, they disagree on whether this implies that these subscriptions therefore should not be terminated, no matter what the budgetary consequences.

There are no competitors which aim at the same part of the market, and the library is a non-profit organization, so performance cannot be measured in terms of profit. The long-term objective of the library is to improve the level of service currently provided to the user, and to look for possibilities to provide new services. As part of the realization of the first objective, organizational measures are taken that aim at making more efficient use of the financial means at the disposal of the library than is done now. These measures are described below.

As part of the realization of the second objective, the possibilities for providing new services are being studied. There is a national EDI network for university libraries and public libraries of which the library is not yet a part. Access to this network would allow the users to find literature fast and request it from the appropriate library anywhere in The Netherlands. Possibilities for extending this kind of service by connecting to a European network are also considered. In addition, the library catalogue should be made available online to users, and there are plans to provide entry into the catalogue by terminals installed at the library itself, through a modem connection, and through the local area network of the university.

To eliminate some problems experienced by the library, and reduce others, the library is reorganized. To spend the library budget more efficiently, all double subscriptions to journals should be terminated, so that for each journal, there is at most one subscription owned by the University. Similarly, books already present in the University library in one department should not be bought by another, unless there is good reason to do so.

Journals cannot be borrowed anymore, neither in single issues nor in bound volumes. All library departments get photocopiers so that papers from journals can be copied without borrowing them.

To reduce theft, documents that can be borrowed are marked in an indelible way, and ports with sensors are installed at the entry of each library.

An IS should be installed that supports library staff in the stricter enforcement of library rules. For example, the borrowing limit of three weeks will be strictly maintained, and to support this enforcement, a report should be produced each week on documents who are borrowed for longer than their allowed lending period, together with a standard letter that is sent to the user. In addition, once every year, all users are to be sent a list of lost or stolen books, so that they become aware of the problem.

Appendix C

An outline of some development methods

C.1 ETHICS

ETHICS (Effective Technical and Human Implementation of Computer-Based Systems) is a sociotechnical system development method, developed by Enid Mumford of the Manchester Business School in the late 1970s and early 1980s [237, 235]. A **sociotechnical** development method is a method to develop a system that consists of of a human subsystem and a technical subsystem. Sociotechnical development is oriented to developing both subsystems in an integrated way, so that the integrated system functions in an optimal way. This development strategy has its background in studies done in the 1950s on the relationship between social structure and technology in organizations [347]. Emery and Trist [95] give a brief introduction to sociotechnical development. Bostrom and Heinen [47] give an introduction to sociotechnical ideas for system developers and apply these ideas to an analysis of success and failure factors of information system development. ETHICS is a sociotechnical development method oriented towards information systems at the operational level of an organization. Pava [256] shows how sociotechnical ideas can be applied to the development of strategic information systems.

ETHICS can best be viewed as a development method for organization units, with equal emphasis on the job satisfaction aspect, the workflow aspect and the information aspect of the unit. Most of the method is spent in requirements determination, which follows a rational choice cycle. Work design and implementation involves designing and implementing a (new or renovated) information system, but there is no specific advice of ETHICS about this.

ETHICS is a participative method that follows the consensus model. The efficiency needs and job satisfaction needs are collected and diagnosed by means of questionnaires given to all people who work in the organization unit. Decisions are made by involving all workers in the unit, who should all support the decisions. The decision about the change option should be verified with the appropriate management authorities.

An outline of the ETHICS method is given in figure C.1.

1. Determine the reasons for change.
2. Determine the boundaries of the organization unit to be developed.
3. Describe the organization unit as it currently is.
4. Define the key objectives of the unit.
5. Define the key tasks performed in the unit.
6. Determine the *information needs* of these tasks.
7. Collect the accidental problems with the current organization of work in the unit, i.e. problems that are due to the way the workflow in the unit is currently implemented. Diagnose these problems. The underlying problems are called *efficiency needs*.
8. Collect information on job satisfaction problems in the current organization of work. Diagnose these, and call the underlying problems *job satisfaction needs*.
9. Use the information needs, efficiency needs and job satisfaction needs to determine the change objectives of the development process.
10. Generate and evaluate organizational change options.
11. Generate and evaluate technical change options.
12. Merge the technical and organizational options and choose one.
13. Make a detailed work design for the chosen option.
14. Implement the work design.
15. Evaluate the new situation.

Figure C.1: Outline of the ETHICS method.

1. **Feasibility study**. Find out if the development of a computer-based system would provide benefits that justify the cost of the development process.
2. **Classical structured analysis**.
   (a) Specify the current system, allowing physical details.
   (b) Specify the current essential system by eliminating all physical details.
   (c) Transform this into an essential model of the desired situation.
   (d) Generate alternative physical implementations of the new system.
   (e) Quantify each alternative by a cost/benefit analysis.
   (f) Choose an option.
   (g) Propose a budget for the chosen option.
   (h) Plan the rest of the project.
   (i) Package the resulting documents into a structured specification.
3. **Structured design**.
   (a) Derive structure charts for the design and add control.
   (b) Design the module structure of the system.
   (c) Package the resulting documents into a structured design specification.
4. **Structured implementation**.

Figure C.2: Outline of classical structured development.

C.2 Structured Development

There are several versions of structured development that can be classified as **classical structured development** and **modern structured development**. In classical development, the current system is reverse engineered to retrieve an essential system model, which is then re-engineered to a model of the desired system. In modern developmemt, the desired system is modeled using event partitioning, without assuming that a current system has been modeled first. Figure C.2 gives an outline of the classical method as proposed by DeMarco [84]. Figure C.3 gives an overview of the modern structured analysis as described by Goldsmith [118].

C.3 SSADM

SSADM (Structured Systems Analysis and Design Method) is the standard method prescribed by the UK government for carrying out development projects for computer-based systems [14, 91, 98]. It assumes that there is a strategic information plan for the business and gives a number of steps that lead from a global information strategy to an implemented information system. SSADM uses structured techniques from a number of other methods, including Entity-Relationship modeling, Structured Analysis and Jackson System

1. Build an essential model of the desired system.
   - 1.1 Build a context diagram.
     - (a) Understand the purpose of the system.
     - (b) Identify the external entities.
     - (c) Define the data flows entering and leaving the system.
     - (d) Check the context diagram.
   - 1.2 Build an event list.
   - 1.3 Build a behavioral model.
     - (a) Build a DFD, an ER diagram, and state transition diagrams (not treated in this volume).
     - (b) Integrate the diagrams.
     - (c) Divide the diagrams into levels.
     - (d) Complete the data dictionary.
     - (e) Add implementation constraints.
2. Build a processor environment model.
   - 2.1 Allocate data transformations and data stores to available processors.
   - 2.2 Document the allocation by means of allocation tables.
3. Specify the human-computer interface.
4. Build a software environment model.
   - 4.1 For each processor, allocate behavior to standard software already available on the processor.
   - 4.2 Allocate remaining behavior to execution units.
   - 4.3 Document the allocation by means of allocation tables.
5. Build a code organization model.
   - 5.1 Translate each DFD into a structure chart.
   - 5.2 Complete the traceability tables that show allocation of transformations to modules.

Figure C.3: Outline of modern structured development.

Development. SSADM assumes that there is a project board which monitors the information strategy of the business and to which the project team executing the SSADM process reports. Every step in the feasibility study is verified with the project board.

SSADM consists of 5 tasks, called *modules*, each of which is divided into one or more tasks called *stages*, which themselves are devided into a sequence of *steps*. Figures C.4 and C.5 give an outline of SSADM. The construction, test and operation tasks are not part of SSADM.

Feasibility study

0. Feasibility

   010 *Prepare feasibility study.* Define scope of the project, identify stakeholders and problem areas, create a high level model of data and activities.

   020 *Define the problem.* Identify the activities and information necessary for the business unit to meet its objectives, identify those aspects of current operations where improvement is required, identify new features of the system, identify nonfunctional requirements.

   030 *Identify feasibility options.* Draw up a list of minimum requirements, to be met by all options. make a list of up to six business system options and up to six technical system options. Combine, trim down to about three, document these and do a cost/benefit analysis and impact analysis of these. Identify preferred option and assist project board and users in selection.

   040 *Assemble feasibility report.*

Requirements analysis

1. Investigate current environment.

   110 *Establish analysis framework.* Review output from feasibility study, identify target users, plan the project.

   120 *Investigate and define requirements.* Investigate current system operation, including frequencies, volumes, etc. Identify intended users and identify opportunities for improving current system, as well as desired functions and data not currently provided.

   130 *Investigate current processing.* Create a data flow diagram of current processing. Identify shortcomings of current processing with users and add to requirements.

   140 *Investigate current data.* Create an entity model of current data. Identify shortcomings of current data with users and add to requirements.

   150 *Derive logical view of current services.* Make logical model of current processing and data.

   160 *Assemble investigation results.*

2. Investigate business system options.

   210 *Define business system options.* Make a list of requirements, define up to six business solutions, cut down to three with user, describe each of these and do a cost/benefit analysis and an impact analysis.

   220 *Select business system option.* Present to project board, record choice made and complete description of this choice.

Figure C.4: Feasibility study and requirements analysis in SSADM.

Requirements specification.

3. Specify requirements.

   310 *Define required system processing*. Transform logical model of current processing to agree with selected option. Define user roles and correlate role with new processing model.

   320 *Develop required data model*. Transform logical model of current data to agree with selected option and with new processing model.

   330 *Derive system functions*. Identify update and enquiry functions, specify I/O interface of each function, cross-reference with user roles and identify critical dialogues.

   340 *Enhance required data model*. Normalize data of selected functions and verify the result with logical data model.

   350 *Develop specification prototyping*. Create prototypes of dialogues, reports, screens and access paths for selected functions. Test with users and iterate if necessary.

   360 *Develop process specification*. Identify for each entity in the logical data model which events create, update or delete it and define a life cycle of events for the entity. Include parallelism, interaction and abnormal termination. Specify all entities affected by an event and define enquiry access paths.

   370 *Confirm system objectives*. Ensure that all functional requirements are met and that all non-functional requirements are defined.

   380 *Assemble requirements specification*.

Logical system specification

4. Investigate technical system options.

   410 *Define technical system options*. Identify ways to implement requirements.

   420 *Select technical system option*.

5. Specify logical design.

   510 *Design user dialogues*.

   520 *Define update processing model*.

   530 *Define enquiry processing model*.

   540 *Assemble logical design*.

Physical design

6. Specify physical design

   610-... (Details omitted)

Figure C.5: Requirements specification, logical system specification and physical design in SSADM.

Bibliography

[1] R.J. Abbott. Program design by informal English descriptions. *Communications of the ACM*, 26:882–894, 1983.

[2] ACM. ACM code of professional conduct. *Communications of the ACM*, 16:262–269, 1973.

[3] P.E. Agre. Book review of Lucy A. Suchman, *Plans and Situated Actions: The Problem of Human-Machine Communication*. *Artificial Intelligence*, 43:369–384, 1990.

[4] W.W. Agresti. The conventional software life-cycle model: its evolution and assumptions. In W.W. Agresti, editor, *New Paradigms for Software Development*, pages 2–5. Computer Society Press, 1986.

[5] W.W. Agresti, editor. *New Paradigms for Software Development*. Computer Society Press, 1986.

[6] B. Alabiso. Transformation of data flow analysis models to object oriented design. In N. Meyrowitz, editor, *Object-Oriented Programming Systems, Languages and Applications, Conference Proceedings*, pages 335–353. ACM Press, 1988. SIGPLAN Notices, volume 23.

[7] M. Alavi. An assessment of the prototyping approach to information systems development. *Communications of the ACM*, 27:556–563, 1984.

[8] M. Alavi and J.C. Wetherbe. Mixing prototyping and data modeling for information system design. *IEEE Software*, 11(5):86–91, May 1991.

[9] R.E. Anderson, D.G. Johnson, D. Gotterbarn, and J. Perrolle. Using the new ACM code of ethics in decision making. *Communications of the ACM*, 36(2):98–105, February 1993.

[10] S.J. Andriole. Fast, cheap requirements: prototype, or else! *IEEE Software*, 14(3):85–87, March 1994.

[11] L.B. Archer. *Technological Innovation — A Methodology*. Inforlink, on behalf of the Science Policy Foundation, 1971.

[12] L.B. Archer. Whatever became of design methodology? *Design Studies*, 1(1):17–18, July 1971.

[13] L.B. Archer. Systematic method for designers. In N. Cross, editor, *Developments in Design Methodology*, pages 57–82. Wiley, 1984. Originally published by *The Design Council*, 1965.

[14] C. Ashworth and M. Goodland. *SSADM: A Practical Approach*. McGraw-Hill, 1990.

[15] M. Asimov. *Introduction to Design*. Prentice-Hall, 1962.

[16] D.E. Avison and A.T. Wood-Harper. *Multiview: An Exploration in Information Systems Development*. Blackwell, 1990.

[17] R.L. Baber. "software engineering" vs. software *engineering*. *Computer*, 22(5):81, 1989.

[18] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990.

[19] S.C. Bailin. An object-oriented requirements specification method. *Communications of the ACM*, 32:608–623, 1989.

[20] J.P. Bansler and K. Bødker. A reappraisal of structured analysis: Design in an organizational context. *ACM Transactions on Information Systems*, 11(2):165–193, April 1993.

[21] R. Barker. *Case*Method: Entity Relationship Modelling*. Addison-Wesley, 1990.

[22] R. Barker. *Case*Method: Tasks and Deliverables*. Addison-Wesley, 1990.

[23] R. Barker and C. Longman. *Case*Method: Function and Process Modelling*. Addison-Wesley, 1992.

[24] V.R. Basili and A.J. Turner. Iterative enhancement: a practical technique for software development. *IEEE Transactions on Software Engineering*, SE-1(4):390–396, December 1975.

[25] C. Batini, S. Ceri, and S.B. Navathe. *Conceptual Database Design: An Entity-Relationship Approach*. Benjamin/Cummings, 1992.

[26] C. Batini and M. Lenzerini. A methodology for data schema integration in the entity relationship model. *IEEE Transactions on Software Engineering*, SE-10:650–664, 1984.

[27] C. Batini, M. Lenzerini, and S.B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.

[28] F.C. Belz. Applying the spiral model: observations on developing system software in ADA. In *Proceedings of the 4th Annual Conference on Ada Technology*, pages 57–66, 1986.

[29] P.L. Berger and T. Luckmann. *The Social Construction of Reality: A Treatise in the Sociology of Knowledge*. Anchor Books, 1984. First edition 1966.

[30] D.M. Berry. Academic legitimacy of the software engineering discipline. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, November 1992. Available through anonymous ftp from ftp.sei.cmu.edu (128.237.2.179).

[31] G. Berry and I. Cosserat. The ESTEREL synchronous programming language and its mathematical semantics. In S. Brookes and G. Winskel, editors, *Seminar on Concurrency*, pages 389–448, 1985. Lecture Notes in Computer Science 197.

[32] L. von Bertalannfy. *General Systems Theory*. Penguin, 1968.

[33] A. Birchenough and J.R. Cameron. JSD and object-oriented design. In J. Cameron, editor, *JSP & JSD - The Jackson Approach to Software Development*, pages 292–304. IEEE Computer Science Press, second edition, 1989.

[34] B.I. Blum. A taxonomy of software development methods. *Communications of the ACM*, 37(11):82–94, November 1994.

[35] B. Boehm and F. Belz. Applying process programming to the spiral model. In *Proceedings of the 4th International Software Process Workshop*, pages 46–56, 1988.

[36] B.W. Boehm. Software engineering. *IEEE Transactions on Computers*, C-25:1226–1241, 1976.

[37] B.W. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[38] B.W. Boehm. Verifying and validating software requirements and design specifications. *IEEE Software*, pages 75–88, January 1984.

[39] B.W. Boehm. A spiral model of development and enhancement. *Software Engineering Notes*, 11(4):14–24, 1986. (Proceedings of International Workshop on Software Process and Software Environments, March 1985).

[40] B.W. Boehm. Improving software productivity. *Computer*, pages 43–57, September 1987.

[41] B.W. Boehm. Implementing risk management. In B. Boehm, editor, *Software Risk Management*, pages 433–440. IEEE Computer Society Press, 1989.

[42] B.W. Boehm. A spiral model of software development and enhancement. *Computer*, pages 61–72, may 1988.

[43] B.W. Boehm, T.E. Gray, and T. Seewalt. Prototyping versus specifying: A multiproject experiment. *IEEE Transactions on Software Engineering*, SE-10:290–302, 1984.

[44] G. Booch. Object-oriented development. *IEEE Transactions on Software Engineering*, SE-12:211–221, 1986.

[45] G. Booch. *Object-Oriented Design with Applications, Second edition.* Benjamin/Cummings, 1994.

[46] A. Borgida, J. Mylopoulos, and H.K.T. Wong. Generalization/specialization as a basis for software specification. In M.L. Brodie, J. Mylopoulos, and J.W. Schmidt, editors, *On Conceptual Modelling*, pages 87–114. Springer, 1984.

[47] R.P. Bostrom and J.S. Heinen. MIS problems and failures: A sociotechnical perspective. Part I: The causes. *MIS Quarterly*, pages 17–32, September 1977.

[48] J.M. Bots, E. van Heck, V. van Swede, and J.L. Simons. *Bestuurlijke Informatiekunde.* Cap Gemini Publishing/Pandata B.V., 1990.

[49] K.E. Boulding. General systems theory — the skeleton of science. *General Systems*, 1:11–17, 1956.

[50] J.W. Brackett. Software requirements: SEI Curriculum Module SEI-CM-19-1.2. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, January 1990. Available through anonymous ftp from ftp.sei.cmu.edu (128.237.2.179).

[51] M.L. Brodie, J. Mylopoulos, and J.W. Schmidt, editors. *On Conceptual Modelling.* Springer, 1984.

[52] M.L. Brodie and E. Silva. Active and passive component modelling: ACM/PCM. In T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, editors, *Information Systems design Methodologies: A Comparitive Review*, pages 41–91. North-Holland, 1982.

[53] F. Brooks. No silver bullet: essence and accidents of software engineering. *Computer*, 20(4):10–19, April 1987.

[54] P.G. Brown. QFD: echoing the voice of the customer. *AT&T Technical Journal*, pages 18–32, March/April 1991.

[55] T.A. Byrd, K.L. Cossick, and R.W. Zmud. A synthesis of research on requirements analysis and knowledge acquisition techniques. *MIS Quarterly*, pages 117–138, March 1992.

[56] J. Cameron, editor. *JSP & JSD - The Jackson Approach to Software Development.* IEEE Computer Science Press, second edition, 1989.

[57] J.R. Cameron. An overview of JSD. *IEEE Transactions on Software Engineering*, SE–12:222–240, 1986.

[58] T.T. Carey and R.E.A. Mason. Information system prototyping: techniques, tools, and methodologies. In B. Boehm, editor, *Software Risk Management*, pages 349–359. IEEE Computer Society Press, 1989. Appeared in *INFOR — The Canadian Journal of Operational Research and Information Processing*, 21(3), 1983, pages 177–191.

[59] E. Carmel, R.D. Whitaker, and J.F. George. PD and Joint Application Design: a transatlantic comparison. *Communications of the ACM*, 36(6):40–48, June 1993.

[60] R. Carnap. *Der logische Aufbau der Welt.* Felix Meiner verlag, 1928.

[61] J.L. Carswell and A.B. Navathe. SA–ER: a methodology that links structured analysis and entity–relationship modeling for database systems. In S. Spaccapietra, editor, *Entity–Relationship Approach*, pages 381–396. North–Holland, 1987.

[62] P. Checkland and J. Scholes. *Soft Systems Methodology in Action*. Wiley, 1990.

[63] P.B. Checkland. *Systems Thinking, Systems Practice*. Wiley, 1981.

[64] P. Chen, editor. *Proceedings of the 1st International Conference on the Entity-Relationship Approach to Systems Analysis and Design*. North-Holland, 1980.

[65] P.P.-S. Chen. The entity-relationship model – Toward a unified view of data. *ACM Transactions on Database Systems*, 1:9–36, 1976.

[66] P.P.-S. Chen. English sentence structure and Entity-Relationship diagrams. *Information Sciences*, 29:127–149, 1983.

[67] C.W. Churchman. *The Systems Approach and Its Enemies*. Basic Books, 1979.

[68] P. Coad and E. Yourdon. *Object-Oriented Analysis*. Yourdon Press/Prentice-Hall, 1990.

[69] E.F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13:377–387, 1970.

[70] E.F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4:397–434, 1979.

[71] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes. *Object-Oriented Development: The FUSION Method*. Prentice-Hall, 1994.

[72] Arthur Young & Company. *The Arthur Young Practical Guide to Information Engineering*. Wiley, 1987.

[73] P.A. Currit, M. Dyer, and H.D. Mills. Certifying the reliability of software. *IEEE Transactions on Software Engineering*, SE-12(1):3–31, 1986.

[74] B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, November 1988.

[75] O.-J. Dahl, E.W. Dijkstra, and C.A.R. Hoare. *Structured Programming*. Academic Press, 1972.

[76] A.M. Davis. A taxonomy for the early stages of the software development life cycle. *The Journal of Systems and Software*, 8:297–311, 1988.

[77] A.M. Davis. *Software Requirements: Objects, Functions, States*. Prentice-Hall, 1993.

[78] A.M. Davis, E.H. Bersoff, and E.R. Comer. A strategy for comparing alternative software development life cycle models. *IEEE Transactions on Software Engineering*, 14:1453–1461, 1988.

[79] A.M. Davis and P.A. Freeman. Guest editor's introduction: Requirements engineering. *IEEE Transactions on Software Engineering*, 17(3):210–211, March 1991.

[80] A.M. Davis and P. Hsia. Giving voice to requirements engineering. *IEEE Software*, 10(6):12–16, March 1994.

[81] G.B. Davis. Strategies for information requirements determination. *IBM Systems Journal*, 21:4–30, 1982.

[82] G.B. Davis and M.H. Olson. *Management Information Systems: Conceptual Foundations, Structure, and Development*. McGraw-Hill, 2nd edition, 1985.

[83] P.A. Dearnley and P.J. Mayhew. In favour of system prototypes and their integration into the system development life cycle. *The Computer Journal*, 26:36–42, 1983.

[84] T. DeMarco. *Structured Analysis and System Specification*. Yourdon Press/Prentice-Hall, 1978.

[85] R. Descartes. Rules for the direction of the mind. In J. Cottingham, R. Stoothoff, and D. Murdoch, editors, *The Philosophical Writings of Descartes*, volume 1. Cambridge University Press, 1985. Trans. D. Murdoch.

[86] J. Dewey. *How We Think: a restatement of the relation of reflective thinking to the educative process*. D.C. Heath and Company, 1933.

[87] E.W. Dijkstra. Notes on structured programming. In *Structured Programming*, pages 1–82. Academic Press, 1972.

[88] J.R. Distaso. Software management — a survey of the practice in 1980s. *Proceedings of the IEEE*, 68(9):1103–1119, September 1980.

[89] M. Dorfman. System and software requirements engineering. In R. Thayer and M. Dorfman, editors, *System and Software Requirements Engineering*, pages 4–16. IEEE Computer Science Press, 1990.

[90] M. Dorfman and R.H. Thayer, editors. *Standards, Guidelines, and Examples on System and Software Requirements Engineering*. Computer Science Press, 1990.

[91] E. Downs, P. Clare, and I. Coe. *Structured Systems Analysis and Design Method: Application and Context*. Prentice-Hall, second edition, 1992.

[92] P. Drucker. *The Practice of Management*. William Heinemann, 1955.

[93] M. Dyer. The management of software engineeering part IV: software development practices. *IBM Systems Journal*, 19(4):451–465, 1980.

[94] R. Elmasri and S.B. Navathe. *Fundamentals of Database Systems*. Benjamin/Cummings, 1989.

[95] F.E. Emery and E.L. Trist. Socio-technical systems. In C.W. Churchman and M. Verhulst, editors, *Management Science, Models and Techniques* Vol. 2, pages 83–97. Pergamon, 1960.

[96] M.D. Ermann, M.B. Williams, and C. Gutierrez, editors. *Compuers, Ethics, and Society*. Oxford University Press, 1990.

[97] G. Ernst and A. Newell. *GPS: A Case Study in Generality and Problem Solving*. Academic Press, 1969.

[98] M. Eva. *SSADM Version 4: A User's guide*. McGraw-Hill, 1992.

[99] M.E. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15:182–211, 1976.

[100] M.E. Fagan. Advances in software inspections. *IEEE Transactions on Software Engineering*, SE-12:744–751, 1986.

[101] R.G. Fichman and C.F. Kemerer. Object-oriented and conventional analysis and design methodologies: Comparison and critique. *Computer*, 25:22–39, October 1992.

[102] S. Fickas and A. Finkelstein, editors. *International Symposium on Requirements Engineering*. IEEE Computer Science Press, 1993.

[103] C. Finkelstein. *An Introduction to Information Engineering*. Addison-Wesley, 1989.

[104] G. Fitzgerald, N. Stokes, and J.R.G. Wood. Feature analysis of contemporary information systems methodologies. *The Computer Journal*, 28:223–230, 1985.

[105] M. Flavin. *Fundamental Concepts of Information Modeling*. Yourdon Press, 1981.

[106] T. Forester and P. Morrison. *Computer Ethics: Cautionary Tales and Ethical Dilemmas in Computing. 2nd edition*. The MIT Press, 1994.

[107] D.P. Freedman and G.M. Weinberg. *Handbook of Walkthroughs, Inspections, and Technical Reviews*. Dorset House, 1990.

[108] C. Gane and T. Sarson. *Structured Systems Analysis: Tools and Techniques*. Prentice-Hall, 1979.

[109] H. Garfinkel. *Studies in Ethnomethodology*. Prentice-Hall, 1967.

[110] O. Gatto. Autosate. *Communications of the ACM*, 7(7):425–432, July 1964.

[111] D.C. Gause and G.M. Weinberg. *Exploring Requirements: Quality Before Design*. Dorset House Publishing, 1989.

[112] D.C. Gause and G.M. Weinberg. *Are Your Lights On?* Dorset House, 1990.

[113] A. Giddens. *Central Problems in Social Theory*. MacMillan, 1979.

[114] T. Gilb. Evolutionary delivery versus the "waterfall model". *ACM Sigsoft Software Engineering Notes*, 10(3):49–61, July 1985.

[115] T. Gilb. *Principles of Software Engineering Management*. Addison-Wesley, 1988.

[116] G.R. Gladden. Stop the life-cycle, I want to get off! *Software Engineering Notes*, 7(2):35–39, April 1982.

[117] J.A. Goguen. Social issues in requirements engineering. In S. Fickas and A. Finkelstein, editors, *International Symposium on Requirements Engineering*, pages 194–195. IEEE Computer Science Press, 1993.

[118] S. Goldsmith. *Real-Time Systems Development*. Prentice-Hall, 1993.

[119] H. Gomaa. A software design method for real-time systems. *Communications of the ACM*, 27(9):938–949, September 1984.

[120] H. Gomaa. Software development of real-time systems. *Communications of the ACM*, 29(7):657–668, July 1986.

[121] H. Gomaa. The impact of prototyping on software system engineering. In R. Thayer and M. Dorfman, editors, *System and Software Requirements Engineering*, pages 543–552. IEEE Computer Science Press, 1990.

[122] H. Gomaa. *Software Design Methods for Concurrent and Real-Time Systems*. Addison-Wesley, 1993.

[123] V.S. Gordon and J.M. Bieman. Rapid prototyping: lessons learned. *IEEE Software*, 12(1):85–94, January 1995.

[124] O. Gotel and A. Finkelstein. Contribution structures. In *Second IEEE International Symposium on Requirements Engineering*. IEEE Computer Society Press, 1995.

[125] F. de Graaf and J.M.A. Berkvens, editors. *Hoofdstukken Informaticarecht*. Samsom H.D. Tjeenk Willink, 1991.

[126] J. Gray. The transaction concept: virtues and limitations. In C. Zaniolo and C. Delobel, editors, *Proceedings of the Seventh International Conference on Very Large Databases*, pages 144–154, Cannes, France, September 9–11 1981.

[127] J.J. van Griethuysen (ed.). Concepts and terminology for the conceptual schema and the information base. Technical Report TC97/SC5/WG3, International Organization of Standards, 1982.

[128] R. Guindon. Designing the design process: exploiting opportunistic thoughts. *Human-Computer Interaction*, 5:304–344, 1990.

[129] O. Gutierrrez. Experimental techniques for information requirements analysis. *Information & Management*, 16:31–43, 1989.

[130] R.D. Hackathorn and J. Karimi. A framework for comparing information engineering methods. *MIS Quarterly*, 12(1):203–220, June 1988.

[131] A.D. Hall. *A Methodology for Systems Engineering*. Van Nostrand, 1962.

[132] A.D. Hall. Three-dimensional morphology of systems engineering. *IEEE Transactions on System Science and Cybernetics*, SSC-5(2):156–160, 1969.

[133] A.D. Hall and R.E. Hagen. Definition of system. In J.A. Litterer, editor, *Organizations: Volume 1, Structure and Behavior*, pages 31–43. Wiley, second edition, 1969.

[134] P. Hall, J. Owlett, and S. Todd. Relations and entities. In G.M. Nijssen, editor, *Modelling in Database Management Systems*, pages 201–220. North-Holland, 1976.

[135] M. Hammer and D. McLeod. Database description with SDM: A semantic database model. *ACM Transactions on Database Systems*, 6:351–386, 1981.

[136] M.Z. Hanani and P. Shoval. A combined methodology for information systems analysis and design based on ISAC and NIAM. *Information Systems*, 11(3):245–253, 1986.

[137] D. Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[138] D. Harel. Biting the silver bullet. *Computer*, 25(1):8–20, Januray 1992.

[139] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. STATEMATE: a working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16:403–414, April 1990.

[140] D. Harel and A. Pnueli. On the development of reactive systems. In K. Apt, editor, *Logics and Models of Concurrent Systems*, pages 477–498. Springer, 1985. NATO ASI Series.

[141] D. Hatley and I. Pirbhai. *Strategies for Real-Time System Specification*. Dorset House, 1987.

[142] J.R. Hauser and D. Clausing. The house of quality. *Harvard Business review*, 66(3):63–73, May–June 1988.

[143] M. Heidegger. *Sein und Zeit*. Max Niemeyer Verlag, 15th edition, 1979. First edition 1927.

[144] S.J. Heims. *John von Neumann and Norbert Wiener: From Mathematics to the Technologies of Life and death*. MIT Press, 1980.

[145] B. Henderson-Sellers and L.L. Constantine. Object-oriented development and functional decomposition. *Journal of Object-Oriented Programming*, pages 11–16, January 1991.

[146] R. Hirschheim and H.K. Klein. Four paradigms of information systems development. *Communications of the ACM*, 32(10):1199–1216, October 1989.

[147] N.R. Howes. On using the users' manual as the requirements specification. In R.H. Thayer, editor, *Software Engineering Project Management*, pages 172–177. IEEE Computer Science Press, 1988.

[148] N.R. Howes. On using the users' manual as the requirements specification II. In R. Thayer and M. Dorfman, editors, *System and Software Requirements Engineering*, pages 164–169. IEEE Computer Science Press, 1990.

[149] P. Hsia, A. Davis, and D. Kung. Status report: requirements engineering. *IEEE Software*, 10(6):75–79, November 1993.

[150] V. Hubka. *Principles of Engineering Design*. Butterworth, 1982. Translated and edited by W.E. Eder.

[151] M.E.C. Hull, A. Zarea-Aliabadi, and D.A. Guthrie. Object-oriented design, Jackson system development (JSD) specifications and concurrency. *Software Engineering Journal*, pages 79–86, March 1989.

[152] W.S. Humphrey. *Managing the Software Process*. Addison-Wesley, 1989.

[153] i Logix. The Languages of STATEMATE. Technical report, i-Logix Inc., 22 Third Avenue, Burlington, Mass. 01803, U.S.A., January 1991.

[154] IEEE. IEEE code of ethics. Technical report, The Institute of Electrical and Electronic Engineers, Inc., 345 East 47th Street, New York, NY 10017-2394, U.S.A., 1979.

[155] D. Ince. Prototyping. In J.A. McDermid, editor, *Software Engineer's Reference Book*, pages 40/1–40/12. Butterworth/Heinemann, 1992.

[156] The Institute of Electrical and Electronic Engineers, Inc., 345 East 47th Street, New York, NY 10017, USA. *Software Engineering Standards*, 1987.

[157] M. Jackson. *Principles of Program Design*. Academic Press, 1975.

[158] M. Jackson. *System Development*. Prentice-Hall, 1983.

[159] M. Jackson. Some complexities in computer-based systems and their implications for system development. In *Proceedings of the 1990 IEEE International Conference on Computer Systems and Software Engineering — COMPEURO'90*, pages 344–351, Tel-Aviv, Israel, 8–10 May 1990. IEEE Computer Society Press.

[160] M. Jackson. Problems, methods and specialization. *IEEE Software*, 11(6):57–62, November 1994.

[161] M. Jackson and P. Zave. Domain descriptions. In S. Fickas and A. Finkelstein, editors, *International Symposium on Requirements Engineering*, pages 56–64. IEEE Computer Science Press, 1993.

[162] M.A. Jackson. Constructive methods in program design. In P. Freeman and A.I. Wasserman, editors, *IEEE Tutorial on Software Design Techniques*, pages 514–532. IEEE Computer Society Press, 1983. Reprinted with permission from *Proceedings of the First Conference of the European Cooperation in Informatics*, vol. 44, 1976, pages 236–262.

[163] I. Jacobson, M. Christerson, P. Johnsson, and G. Övergaard. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Prentice-Hall, 1992.

[164] P. Jalote. Functional refinement and nested objects for object–oriented design. *IEEE Transactions on Software Engineering*, 15(3):264–270, March 1989.

[165] R.W. Jensen and C.C. Tonies. *Software Engineering*. Prentice-Hall, 1979.

[166] JMA. Information Engineering Methodology/Facility (IEM/IEF) Overview Seminar, 1990. Course notes.

[167] D.G. Johnson. *Computer Ethics*. Prentice-Hall, 1985.

[168] J.C. Jones. *Design Methods: Seeds of Human Futures*. Wiley, 1970.

[169] J.C. Jones. A method for systematic design. In N. Cross, editor, *Developments in Design Methodology*, pages 9–31. Wiley, 1984. Originally published in J.C. Jones, D. Thornley (eds.), *Conference on Design Methods*, Pergamon, 1963.

[170] D. Katz and R.L. Kahn. *The Social Psychology of Organizations*. Wiley, 1978. Second edition.

[171] P.G.W. Keen and M.S. Scott Morton. *Decision Support Systems: An Organizational Perspective*. Addison-Wesley, 1978.

[172] S.E. Keller, L.G. Kahn, and R.B. Panara. Specifying software quality requirements with metrics. In R. Thayer and M. Dorfman, editors, *System and Software Requirements Engineering*, pages 145–163. IEEE Computer Science Press, 1990.

[173] J.G. Kemeny. *A Philosopher Looks at Science*. Van Nostrand, 1959.

[174] K.E. Kendall and J.E. Kendall. *Systems Analysis and Design*. Prentice-Hall, 1992. Second edition.

[175] W. Kent. A simple guide to five normal forms in relational database theory. *Communications of the ACM*, 26(2):120–125, 1983.

[176] W. Kent. The breakdown of the information model in MDBs. *Sigmod record*, 20(4):10–15, December 1991.

[177] W. Kent. A rigorous model of object reference, identity, and existence. *Journal of Object-Oriented Programming*, 4(3):28–36, June 1991.

[178] S.N. Khoshafian and G.P. Copeland. Object identity. In *Object-Oriented Programming Systems, Languages and Applications*, pages 406–416, 1986. SIGPLAN Notices 22 (12).

[179] R. King and D. McLeod. A unified model and methodology for conceptual database design. In M.L. Brodie, J. Mylopoulos, and J.W. Schmidt, editors, *On Conceptual Modelling*, pages 313–327. Springer, 1984.

[180] R. King and D. McLeod. Semantic data models. In S. Yao, editor, *Principles of Database Design*, pages 115–150. Prentice-Hall, 1985.

[181] W. Kneale and M. Kneale. *The Development of Logic*. Clarendon Press, 1962.

[182] B.V. Koen. Toward a definition of the engineering method. *Engineering Education*, 75:150–155, December 1984.

[183] B.V. Koen. Definition of the engineering method. 1985.

[184] D.A. Kolb and A.L. Frohman. An organization development approach to consulting. *Sloan Management review*, 12(1):51–65, Fall 1970.

[185] J.A. Kowal. *Analyzing Systems*. Prentice-Hall, 1988.

[186] T. Kuhn. *The Structure of Scientific Revolutions*. University of Chicago Press, second, enlarged edition edition, 1970.

[187] I. Lakatos. Falsification and the methodology of scientific research programmes. In I. Lakatos and A. Musgrave, editors, *Criticism and the Growth of Knowledge*, pages 91–196. Cambridge University Press, 1970.

[188] F. Land and R. Hirschheim. Participative systems design: Rationale, tools and techniques. *Journal of Applied Systems Analysis*, 10:91–107, 1983.

[189] J.Z. Lavi, A. Agrawala, R. Buhr, K. Jackson, M. Jackson, and B. Lang. Computer based systems engineering workshop. In J.E. Tomayko, editor, *Software Engineering Education*, pages 149–163. Springer, 1990. Lecture Notes in Computer Science 536.

[190] H.W. Lawson. Philosophies for engineering computer–based systems. *Computer*, 23(12):52–63, December 1990.

[191] M.M. Lehman. Software engineering, the software process and their support. *Software Engineering Journal*, 5(6):243–258, September 1991.

[192] S.W. Liddle, D.W. Embley, and A.N. Woodfield. Cardinality constraints in semantic data models. *Data and Knowledge Engineering*, 11:235–270, 1993.

[193] G.E. Lindblom. The science of 'muddling through'. *Public Administration Review*, 19:79–88, 1959.

[194] O.I. Lindland, G. Sindre, and A. Sølvberg. Understanding quality in conceptual modeling. *IEEE Software*, 11(2):42–49, March 1994.

[195] R.C. Linger. The management of software engineeering part III: Software design practices. *IBM Systems Journal*, 19(4):432–450, 1980.

[196] R.C. Linger. Cleanroom process model. *IEEE Software*, 11:50–58, March 1994.

[197] A. Lopes and F. Costa. Rewriting for reuse. In *Proceedings ERCIM Workshop, Nancy, November 2-4*, pages 43–55. INRIA, 1993.

[198] P. Loucopoulos and V. Karakostas. *System Requirements Engineering*. McGraw-Hill, 1995.

[199] M. Lubars, C. Potts, and C. Richter. A review of the state of the practice in requirements modeling. In S. Fickas and A. Finkelstein, editors, *International Symposium on Requirements Engineering*, pages 2–14. IEEE Computer Science Press, 1993.

[200] M. Lundeberg. The ISAC approach to specification of information systems and its application to the organization of an IFIP working conference. In T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, editors, *Information Systems Design Methodologies: A Comparative Review*, pages 173–234. North-Holland, 1982.

[201] M. Lundeberg. An approach for involving the users in the specification of information systems. In P. Freeman and A.I. Wasserman, editors, *IEEE Tutorial on Software Design Techniques*, pages 133–155. IEEE Computer Society Press, 1983. Reprinted with permission from *Formal Models and Practical Tools for Information Systems Design*, H.-J. Schneider (ed.), North-Holland 1979.

[202] M. Lundeberg, G. Goldkuhl, and A. Nilsson. A systematic approach to information systems development - I. Introduction. *Information Systems*, 4:1–12, 1979.

[203] M. Lundeberg, G. Goldkuhl, and A. Nilsson. A systematic approach to information systems development - II. Problem and data oriented methodology. *Information Systems*, 4:93–118, 1979.

[204] M. Lundeberg, G. Goldkuhl, and A. Nilsson. *Information Systems Development — A Systematic Approach*. Prentice-Hall, 1981.

[205] Luqi and W. Royce. Status report: computer-aided prototyping. *Computer*, pages 77–81, November 1992.

[206] R.A. MacKenzie. The management process in 3–D. In R.H. Thayer, editor, *Software Engineering Project Management*, pages 11–14. IEEE Computer Science Press, 1988. First appeared in *Harvard Business Review*, November/December 1969.

[207] B.J. MacLennan. Values and objects in programming languages. *Sigplan Notices*, 17(12):70–79, December 1982.

[208] I. Mangham. *The Politics of Organizational Change*. Associated Business Press, 1979.

[209] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent System Specification*. Springer, 1992.

[210] D.A. Marca and C.L. Gowan. *SADT: Structured Analysis and Design Technique*. McGraw-Hill, 1988.

[211] J.G. March and R. Weissinger-Baylon, editors. *Ambiguity and Command: Organizational Perspectives on Military decision Making*. Pitman, 1986.

[212] J. Martin. *Strategic Data-Planning Methodologies*. Prentice-Hall, 1982.

[213] J. Martin. *Information Engineering*. Prentice-Hall, 1989. Three volumes.

[214] J. Martin. *Information Engineering, Book I: Introduction*. Prentice-Hall, 1989.

[215] J. Martin. *Information Engineering, Book II: Planning and analysis*. Prentice-Hall, 1989.

[216] J. Martin. *Information Engineering, Book III: Design and construction*. Prentice-Hall, 1989.

[217] J. Martin and C. Finkelstein. *Information Engineering*. Savant Institute, 2 New Street, Carnforth, Lancashire, LA5 9BX, England, 1981. Two volumes.

[218] J. Martin and J.J. Odell. *Object-Oriented Analysis and Design*. Prentice-Hall, 1992.

[219] P.C. Masiero and F.S.R. Germano. JSD as object oriented design method. *Software Engineering Notes*, 13(3):22–23, July 1988.

[220] J.A. McCall, P.K. Richards, and G.F. Walters. Factors in software quality assurance. Technical Report RADC-TR-77-369, Rome Air Development Center, 1977.

[221] D.D. McCracken and M.A. Jackson. A minority dissenting position. In W.W. Cotterman, J.D. Couger, B.L. Enger, and F. Harold, editors, *Systems Analysis and Design — A Foundation for the 1980's*, pages 551–553. Elsevier - North-Holland, 1981.

[222] J. McDermid and P. Rook. Software development process models. In J.A. McDermid, editor, *Software Engineer's Reference Book*, pages 15/1–15/36. Butterworth/Heinemann, 1992.

[223] F.W. McFarlan. Portfolio approach to information systems. In B. Boehm, editor, *Software Risk Management*, pages 17–25. IEEE Computer Society Press, 1989. Appeared in *Harvard Business review*, January/February 1974.

[224] J.D. McGregor and D.M. Dyer. Inheritance and state machines. *Software Engineering Notes*, 18(4):61–69, 1993.

[225] S.M. McMenamin and J.F. Palmer. *Essential Systems Analysis*. Yourdon Press/Prentice Hall, 1984.

[226] A.T. McNeille. Jackson system development (JSD). In T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, editors, *Information Systems Design Methodologies: Improving the Practice*. North-Holland, 1986.

[227] S.J. Mellor and P.T. Ward. *Structured Development for Real-Time Systems*. Prentice-Hall/Yourdon Press, 1985. Volume 3: Implementation Modeling Techniques.

[228] Michael Jackson Limited. *JSD Course Notes*, 1986.

[229] G.A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63:81–97, March 1956.

[230] J.C. Miller. Conceptual models for determining information requirements. In *AFIPS Conference proceedings*, volume 25, pages 609–620, 1964. Spring Joint Conference.

[231] R. Milner. *A Calculus of Communicating Systems*. Springer, 1980. Lecture Notes in Computer Science 92.

[232] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[233] M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

[234] H. Mintzberg, D. Raisinghani, and A. Théorêt. The structure of "unstructured" decision processes. *Administrative Science Quarterly*, 21, June 1976.

[235] E. Mumford. *Designing Human Systems for New Technology: The ETHICS Method*. Manchester Business School, 1983.

[236] E. Mumford. Participation from Aristotle to today. In Th. Bemelmans, editor, *Beyond Productivity: Information Systems Development for Organizational Effectiveness*, pages 95–105. North-Holland, 1984.

[237] E. Mumford and M. Weir. *Computer Systems in Work Design – The ETHICS Method*. Associated Business Press, 1979.

[238] G. Nadler. An investigation of design methodology. *Management Science*, 13(10):B–642–B–655, June 1967.

[239] E. Nagel. *The Structure of Science*. Routledge and Kegan Paul, 1961.

[240] National Computing Center, Oxford road, Manchester, M1 7ED, U.K. *The STARTS Guide: A Guide to Methods and Software Tools for the Construction of Large Real-Time Systems*, 1987. Prepared by industry with the support of the DTI and NCC.

[241] J.D. Naumann, G.B. Davis, and J.D. McKeen. Determining information requirements: a contingency method for selection of a requirements assurance strategy. *Journal of System and Software Sciences*, 1:273–281, 1980.

[242] S. Navathe, R. Elmasri, and J. Larson. Integrating user views in database design. *Computer*, pages 50–62, January 1986.

[243] G.M. Nijssen, editor. *Modelling in Database Management Systems*. North-Holland, 1976.

[244] G.M. Nijssen and T.A. Halpin. *Conceptual Schema and Relational Database Design*. Prentice-Hall, 1989.

[245] H. Obbink. Systems engineering environments of ATMOSPHERE. In A. Endres and H. Webers, editors, *Software Development Environments and CASE Technology*, pages 1–17. Springer, 1991. Lecture Notes in Computer Science 509.

[246] T.W. Olle, J. Hagelstein, I.G. Macdonald, C. Rolland, H.G. Sol, F.J.M. van Assche, and A.A. Verrijn-Stuart. *Information Systems Methodologies: A Framework for Understanding*. Addison-Wesley, 1988.

[247] T.W. Olle, H.G. Sol, and C.J. Tully, editors. *Information Systems Design Methodologies: A Feature Analysis*. North-Holland, 1983.

[248] T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, editors. *Information System Design Methodologies: A Comparative Review*. North-Holland, 1982.

[249] T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, editors. *Information Systems Design Methodologies: Improving the Practice*. North-Holland, 1986.

[250] M.A. Ould. Quality control and assurance. In J.A. McDermid, editor, *Software Engineer's Reference Book*, pages 29/1–29/12. Butterworth/Heinemann, 1992.

[251] M. Page-Jones. *The Practical Guide to Structured Systems Design*. Prentice-Hall, 2nd edition, 1988.

[252] M. Page-Jones. Comparing techniques by means of encapsulation and connascance. *Communications of the ACM*, 35(9):147–151, September 1990.

[253] D.L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 5:1053–1058, 1972.

[254] D.L. Parnas. Software aspects of strategic defense systems. *Communications of the ACM*, 28(12):1326–1335, December 1985.

[255] D.L. Parnas and P.C. Clements. A rational design process: How and why to fake it. *IEEE Transactions on Software Engineering*, SE-12:251–257, 1986.

[256] C. Pava. *Managing New Office Technology: An Organizational Strategy*. The Free Press, 1983.

[257] L.J. Peters. *Software Design: Methods and Techniques*. Prentice-Hall, 1981.

[258] L.J. Peters and L.L. Tripp. A model of software engineering. In *Third International Conference on Software Engineering*, pages 63–70. IEEE Computer Science Press, 1978.

[259] K. Pohl. The three dimensions of requirements engineering: a framework and its applications. *Information Systems*, 19(3):243–258, 1994.

[260] M. Polanyi. *Personal Knowledge*. Chicago University Press, 1958.

[261] M. Polanyi. *The Tacit Dimension*. Routledge and Kegan Paul, 1966.

[262] M. Pollner. *Mundane Reason: Reality in Everyday and Sociological Discourse*. Cambridge University Press, 1987.

[263] G. Polya. *How to Solve it. A New Aspect of Mathematical Method*. Princeton University Press, second edition, 1985. First edition 1945.

[264] K.R. Popper. *The Logic of Scientific Discovery*. Hutchinson, 1959.

[265] C. Potts and G. Bruns. Recording the reasons for design decisions. In *10th International Conference on Software Engineering*, pages 418–427, 1988.

[266] Euromethod project. Euromethod case study. Technical Report EM-2.5-CS, Euromethod project, 1994.

[267] Euromethod project. Euromethod concepts manual 1: Transaction model. Technical Report EM-3.1-TM, Euromethod project, 1994.

[268] Euromethod project. Euromethod concepts manual 2: Deliverable model. Technical Report EM-3.2-DM, Euromethod project, 1994.

[269] Euromethod project. Euromethod concepts manual 3: Strategy model. Technical Report EM-3.3-SM, Euromethod project, 1994.

[270] Euromethod project. Euromethod customer guide. Technical Report EM-2.1-CG, Euromethod project, 1994.

[271] Euromethod project. Euromethod delivery planning guide. Technical Report EM-2.3-DPG, Euromethod project, 1994.

[272] Euromethod project. Euromethod dictionary. Technical Report EM-4-ED, Euromethod project, 1994.

[273] Euromethod project. Euromethod method bridging guide. Technical Report EM-2.4-MBG, Euromethod project, 1994.

[274] Euromethod project. Euromethod overview. Technical Report EM-1-EO, Euromethod project, c/o Sema Group, 16, rue Barbès, 92126 Montrouge Cedex, France, 1994.

[275] Euromethod project. Euromethod supplier guide. Technical Report EM-2.2-SG, Euromethod project, 1994.

[276] S. Pugh. *Integrated Product Engineering*. Addison-Wesley, 1991.

[277] H. Reichenbach. *Experience and Prediction*. University of Chicago press, 1938.

[278] H. Reichenbach. *Elements of Symbolic Logic*. The Free Press/Collier-MacMillan, 1947.

[279] H.W.J. Rittel and M.M. Webber. Planning problems are wicked problems. In N. Cross, editor, *Developments in Design Methodology*, pages 135–144. Wiley, 1984. Originally published as part of "Dilemmas in a general theory of planning" in *Policy Sciences*, 4 (1973), 155–169.

[280] R. Rock-Evans. *Analysis within the Systems Development Life Cycle*, volume 1: Data Analysis —The Deliverables. Pergamon Infotech, 1987.

[281] R. Rock-Evans. *Analysis within the Systems Development Life Cycle*, volume 2: Data Analysis —The Methods. Pergamon Infotech, 1987.

[282] R. Rock-Evans. *Analysis within the Systems Development Life Cycle*, volume 3: Activity Analysis —The Deliverables. Pergamon Infotech, 1987.

[283] R. Rock-Evans. *Analysis within the Systems Development Life Cycle*, volume 4: Activity Analysis —The Methods. Pergamon Infotech, 1987.

[284] R. Rock-Evans. *A Simple Introduction to Data and Activity Analysis*. Computer Weekly Publications, 1989.

[285] J.F. Rockart. Chief executives define their own data needs. *Harvard Business Review*, 57(2):81–93, April/May 1979.

[286] G.-C Roman, M.J. Stucki, W.E. Ball, and W.D. Gillett. A total system design framework. *Computer*, 17(5):15–26, May 1984.

[287] P. Rook. Controlling software projects. In R.H. Thayer, editor, *Software Engineering Project Management*, pages 108–117. IEEE Computer Science Press, 1988. Appeared in the *Software Engineering Journal*, January 1986.

[288] P. Rook. Project planning and control. In J.A. McDermid, editor, *Software Engineer's Reference Book*, pages 27/1–27/36. Butterworth/Heinemann, 1992.

[289] N.F.M. Roozenburg and J. Eekels. *Productontwerpen, Struktuur en Methoden*. Lemma B.V., 1991.

[290] D.T. Ross. Douglass Ross talks about structured analysis. *Computer*, 18(7):80–88, July 1085.

[291] D.T. Ross. Structured analysis (SA): A language for communicating ideas. *IEEE Transactions on Software Engineering*, SE-3(1):16–34, January 1977.

[292] D.T. Ross. Applications and extensions of SADT. *Computer*, 18(4):25–34, April 1985.

[293] D.T. Ross and J.W. Brackett. An approach to structured analysis. *Computer Decisions*, 8(9):40–44, September 1976.

[294] D.T. Ross and K.E. Schoman. Structured analysis for requirements definition. *IEEE Transactions on Software Engineering*, SE-3(5):6–15, January 1977.

[295] W.W. Royce. Managing the development of large software systems. In R.H. Thayer, editor, *Software Engineering Project Management*, pages 118–127. IEEE Computer Science Press, 1988. Appeared in *Proceedings of IEEE WESCON* 1970, IEEE, pp. 1–9.

[296] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-oriented modeling and design*. Prentice-Hall, 1991.

[297] H.D. Ruys. *ISAC voor docenten*. Moret Ernst & Young, 1990.

[298] G. Saake, P. Hartel, R. Jungclaus, R.J. Wieringa, and R.B. Feenstra. Inheritance conditions for object life cycle diagrams. In U.W. Lipeck and G. Vossen, editors, *Formale Grundlagen für den Entwurf von Informationsystemen*, pages 79–88. Institut für Informatik, Universität Hannover, Postfach 6009, D-30060, Hannover, May 1994. Informatik-Berichte Nr. 03/94.

[299] M. Saeki, H. Horai, and H. Enomoto. Software development process from natural language specification. In *11th International Conference on Software Engineering*, pages 64–73. IEEE Computer Society press, May 15–18 1989.

[300] B. Sanden. Systems programming with JSP: example — a VDU controller. *Communications of the ACM*, 28(10):1059–1067, October 1985. See also the exchange in the *Communications of the ACM*, 29(2), February 1986, 89–90.

[301] B. Sanden. The case for eclectic design of real-time software. *IEEE Transactions on Software Engineering*, 15(3), March 1989.

[302] B. Sanden. An entity-life modeling approach to the design of concurrent software. *Communications of the ACM*, 32(3):330–343, March 1989.

[303] B. Sanden. *Software Systems Construction with Examples in ADA*. Prentice-Hall, 1994.

[304] E. Seidewitz. General object-oriented software development: Background and experience. *The Journal of Systems and Software*, 9:95–108, 1989.

[305] E. Seidewitz and M. Stark. Toward a general object-oriented software development methodology. *ADA Letters*, 7(4):54–67, july/august 1987.

[306] R.W. Selby, V.R. Basili, and F.T. Baker. Cleanroom software development: an empirical evaluation. *IEEE Transactions on Software Engineering*, SE-13(9):1027–1037, September 1987.

[307] B. Selic, G. Gullekson, and P.T. Ward. *Real-Time Object-Oriented Modeling*. Wiley, 1994.

[308] S. Shlaer and S.J. Mellor. *Object-Oriented Systems Analysis: Modeling the World in Data*. Prentice-Hall, 1988.

[309] S. Shlaer and S.J. Mellor. *Object Lifecycles: Modeling the World in States*. Prentice-Hall, 1992.

[310] J. Shomenta, G. Kamp, B. Hanson, and B. Simpson. The application approach worksheet: an evaluative tool for matching new development methods with appropriate applications. *MIS Quarterly*, 7(4):1–10, 1983.

[311] P. Shoval. An integrated methodology for functional analysis, process design and database design. *Information Systems*, 16(1):49–64, 1991.

[312] K. Shumate. Structured analysis and object–oriented design are compatible. *Ada Letters*, 11(4):78–90, May/June 1991.

[313] K. Shumate and M. Keller. *Software Specification and Design: A Disciplined Approach for Real-Time Systems*. Wiley, 1992.

[314] H. Simon. The architecture of complexity. *Proceedings of the Aristotelian Society*, 106:467–482, 1962. Reprinted in H. Simon, *The Sciences of the Artificial*, Second edition, MIT Press, 1961.

[315] H.A. Simon. A behavioral model of rational choice. *Quarterly Journal of Economics*, 69:99–118, 1955.

[316] H.A. Simon. *The New Science of Management Decision*. Harper and Row, 1960.

[317] H.A. Simon. On the concept of organizational goal. *Administrative Science Quarterly*, 9:1–22, 1964.

[318] H.A. Simon. *The Sciences of the Artificial*. MIT Press, 1969.

[319] J.L. Simons and G.M.A. Verheijen. *Informatiestrategie als Managementsopgave: Planning, Ontwikkeling en Beheer van de Informatieverzorging op Basis van Information Engineering*. Kluwer Bedrijfswetenschappen, 1991.

[320] J. M. Smith and D.C.P. Smith. Database abstractions: Aggregation and generalization. *ACM Transactions on Database Systems*, 2:105–133, 1977.

[321] J.M. Smith and D.C.P. Smith. Database abstractions: Aggregation. *Communications of the ACM*, 20:405–413, June 1977.

[322] K. Southwell. Managing software engineering teams. In J.A. McDermid, editor, *Software Engineer's Reference Book*, page Chapter 32. Butterworth/Heinemann, 1992.

[323] J.F. Sowa and J.A. Zachman. Extending and formalizing the framework for information systems architecture. *IBM Systems Journal*, 31(3):590–616, 1992.

[324] R.A. Sprague, K.J. Singh, and R.T. Wood. Concurrent engineering in product development. *IEEE Design and Test of Computers*, 8(1):6–13, March 1991.

[325] W. Stevens, G. Myers, and L. Constantine. Structured design. *IBM Systems Journal*, 13:115–139, 1974.

[326] D.A. Stokes. Requirements analysis. In J.A. McDermid, editor, *Software Engineer's Reference Book*, pages 16/1–16/21. Butterworth/Heinemann, 1992.

[327] V.C. Storey. Relational database design based on the Entity-Relationship model. *Data and Knowledge Engineering*, 7(1):47–83, June 1991.

[328] V.C. Storey and R.C. Goldstein. A methodology for creating user views in database designs. *ACM Transactions on Database Systems*, 13(3):305–338, September 1988.

[329] P.J. van Strien. *Praktijk als Wetenschap. Methodologie van het Sociaal-Wetenschappelijk Handelen*. Van Gorcum, 1986.

[330] L. Suchman and E. Wynn. Procedures and problems in the office. *Office: Technology and People*, 2:135–154, 1984.

[331] L.A. Suchman. Office procedures as practical action: Models of work and system design. *ACM Transactions on Office Information Systems*, 1:320–328, 1983.

[332] A. Sutcliffe. *Jackson System Development*. Prentice-Hall, 1988.

[333] A.G. Sutcliffe. Object-oriented systems development: survey of structured methods. *Information and Software technology*, 33(6):433–442, August 1991.

[334] W. Swartout and R. Balzer. On the inevitable intertwining of specification and implementation. *Communications of the ACM*, 25:438–440, 1982.

[335] V. van Swede and J.C. van Vliet. A flexible framework for contingent information system modelling. *Information and Software Technology*, 35(9):530–548, September 1993.

[336] W.M. Taggart, Jr. and M.O. Tharp. A survey of information requirements analysis techniques. *ACM Computing Surveys*, 9:273–290, 1977.

[337] A.S. Tanenbaum. *Structured Computer Organization*. Prentice-Hall, 3rd edition, 1990.

[338] T.J. Teory. *Database Modeling and Design: The Entity-Relationship Approach*. Morgan Kaufmann, 1990.

[339] T.J. Teory, D. Yang, and J.P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys*, 18:197–222, 1986.

[340] B. Thalheim. Fundamentals of cardinality constraints. In G. Pernul and A.M. Tjoa, editors, *Entity-Relationship Approach –ER'92*, pages 7–23. Springer, 1992. Lecture Notes in Computer Science 645.

[341] R.H. Thayer, editor. *Software Engineering Project Management*. IEEE Computer Science Press, 1988.

[342] R.H. Thayer. Software engineering project management: A top-down view. In R.H. Thayer, editor, *Software Engineering Project Management*, pages 15–53. IEEE Computer Science Press, 1988.

[343] R.H. Thayer and M. Dorfman, editors. *System and Software Requirements Engineering*. IEEE Computer Science Press, 1990.

[344] R.H. Thayer and A.B. Pyster. Guest editorial: software engineering project management. *IEEE Transactions on Software Engineering*, SE-10(1):2–3, January 1984.

[345] B. Thomé, editor. *Systems Engineering: Principles and Practice of Computer-Based Systems Engineering*. Wiley, 1993.

[346] B. Thomeé. Definition and scope of systems engineering. In B. Thomé, editor, *Systems Engineering: Principles and Practice of Computer-Based Systems Engineering*, pages 1–23. Wiley, 1993.

[347] E.L. Trist and K.W. Bamforth. Some social and psychological consequences of the Longwall method of coal-getting. *Human Relations*, 4:3–38, 1951.

[348] C. Tully. System development activity. In B. Thomé, editor, *Systems Engineering: Principles and Practice of Computer–Based Systems Engineering*, pages 45–80. Wiley, 1993.

[349] J. in't Veld. *Analyse van Organisatieproblemen: Een Toepassing van Denken in Systemen en Processen,* vijfde druk. Stenfert Kroese, 1988.

[350] J. Vincent, A. Waters, and J. Sinclair. *Software Quality Assurance. Volume 1: Practice and Implementation*. Prentice-Hall, 1988.

[351] W. Visser. More or less following a plan during design: opportunistic deviations in specification. *International Journal of Man–Machine Studies*, 33:247–278, 1990.

[352] P.T. Ward. The transformation schema: An extension of the data flow diagram to represent control and timing. *IEEE Transactions on Software Engineering*, SE-12:198–210, 1986.

[353] P.T. Ward. How to integrate object orientation with structured analysis and design. *Computer*, pages 74–82, March 1989.

[354] P.T. Ward and S.J. Mellor. *Structured Development for Real-Time Systems*. Prentice-Hall/Yourdon Press, 1985. Volume 1: Introduction and Tools.

[355] P.T. Ward and S.J. Mellor. *Structured Development for Real-Time Systems*. Prentice-Hall/Yourdon Press, 1985. Volume 2: Essential Modeling Techniques.

[356] *Webster's Dictionary of English Usage*, 1989.

[357] G.M. Weinberg. *An Introduction to General Systems Thinking*. Wiley, 1975.

[358] V. Weinberg. *Structured Analysis*. Yourdon Press, 1978.

[359] M. West. Quality function deployment. In *Colloquium on Tools and Techniques for Maintaining Traceability During Design*, Savoy Place, London WC2R OBL, U.K., 2 December 1991.

[360] S. White, M. Alford, J. Holtzman, S. Kuehl, B. McCay, D. Oliver, D. Owens, C. Tully, and A. Willey. Systems engineering of computer-based systems. *Computer*, 26(11):54–65, November 1993.

[361] N. Wiener. *Cybernetics*. MIT Press, 1965.

[362] R.J. Wieringa. Three roles of conceptual models in information system design and use. In E.D. Falkenberg P. Lindgreen, editor, *Information System Concepts: An In-Depth Analysis*, pages 31–51. North-Holland, 1989.

[363] R.J. Wieringa. Object-oriented analysis, structured analysis, and Jackson System Development. In F. van Assche, B. Moulin, and C. Rolland, editors, *Object Oriented Approach in Information Systems*, pages 1–21. North-Holland, 1991.

[364] R.J. Wieringa. Combining static and dynamic modeling methods: a comparison of four methods. *The Computer Journal*, 38(1):17–30, 1995.

[365] R.J. Wieringa and W. de Jonge. Object identifiers, keys, and surrogates. *Theory and Practice of Object Systems*, 1(2):101–114, 1995.

[366] R.J. Wieringa, J.-J. Ch. Meyer, and H. Weigand. Specifying dynamic and deontic integrity constraints. *Data and Knowledge Engineering*, 4:157–189, 1989.

[367] R.D. Williams. Managing the development of reliable software. In *Proceedings of the 1975 International Conference on Reliable Software*, pages 3–8, April 1975.

[368] T. Winograd. What does it mean to understand a language? In D.A. Norman, editor, *Perspectives on Cognitive Science*, pages 231–263. Ablex, 1981.

[369] T. Winograd and F. Flores. *Understanding Computers and Cognition: A New Foundation for Design*. Ablex, 1986.

[370] J.G. Wolff. The management of the spiral model: 'Project SP' and the 'new spiral model'. In B. Boehm, editor, *Software Risk Management*, pages 481–491. IEEE Computer Society Press, 1989. Appeared in *Software Engineering Journal*, May 1989.

[371] D.P. Wood and W.G. Wood. Comparative evaluations of specification methods for real-time systems. Technical Report CMU/SEI-89-TR-36 ADA219187, Software Engineering Institute, Carnegie Mellon University Pittsburgh, PA 15213-3890, 1989.

[372] S. Wrycza. The ISAC-driven transition between requirements analysis and ER conceptual modeling. *Information Systems*, 15(6):603–614, 1990.

[373] R.T. Yeh and P.A. Ng. Software requirements — a management perspective. In R. Thayer and M. Dorfman, editors, *System and Software Requirements Engineering*, pages 450–641. IEEE Computer Science Press, 1990.

[374] R.T. Yeh, P. Zave, A.P. Conn, and G.E. Cole. Software requirements: new directions and perspectives. In C.R. Vick and C.V. Ramamoorthy, editors, *Handbook of Software Engineering*, pages 519–543. Van Nostrand Reinhold Company, 1984.

[375] E. Yourdon. *Structured Walkthroughs*. Prentice-Hall, 1985.

[376] E. Yourdon. *Modern Structured Analysis*. Prentice-Hall, 1989.

[377] E. Yourdon. *Object-Oriented Systems Design: An Integrated Approach*. Prentice-Hall, 1994.

[378] E. Yourdon and L.L. Constantine. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Prentice-Hall, 1979.

[379] J.A. Zachman. A framework for information systems architecture. *IBM Systems Journal*, pages 276–292, 1987.

[380] P. Zave and D. Jackson. Practical specification techniques for control-oriented systems. In G.X. Ritter, editor, *Information Processing 89*, pages 83–88. North-Holland, 1989.

[381] P. Zave and M. Jackson. Conjunction as composition. *ACM Transactions on Software Engineering and Methodology*, 2(4):379–411, October 1993.

[382] M.V. Zelkowitz. Resource utilization during software development. *Journal of Systems and Software*, 8(4):331–336, 1988.

Index