How to contribute to the joint course on software engineering by case studies

Zoran Budimac, Klaus Bothe

Version: February 27, 2004.

Abstract. This document describes where and how the joint course on software engineering [1] uses case studies in the course and in assignments. Therefore, this document also serves as an instruction on how to build new case studies (which activities should be performed and which documents produced) in order to successfully replace the existing case studies.

1 Introduction

The Joint Course on Software Engineering (JCSE) [1] currently uses the following case studies throughout the course and in some assignments:

- the main one, used in 13 topics of the course,
- the supporting one, used in 4 topics of the course,
- additional ones, used in individual topics and in assignments.

There are always on-going efforts to include these case studies in even more topics and also to use additional case studies in some topics.

Currently,

- the main case study is ‘Seminar Organization’, taken from [2]
- the supporting one is ‘XCTL’
- additional ones are local (i.e. are not used in more than one topic or assignment, but still pretty important) and will not be separately discussed in this text.

Topics	**Main** (the number of slides	**Supporting** (the number of slides	**Additional** (the name of the example)
1. What is software engineering			
2. Quality criteria for software products			
3. Software process models			
4. Basic concepts for software development documents			
5. Results of the “analysis and definition” phase	**24**	**1**	
6. Cost estimation	**20**		
7. Function-oriented view	**12**	**2**	
8. Data-oriented view	**5**		
9. Rule-oriented view	**8**		Policy in paying off checks (**8** slides)
10. Structured analysis	**19**		
11. State-oriented view	**5**		Setting of the digital watch (**7** slides)
12. Scenario-oriented view	**4**		
13. Object-oriented analysis	**13**		
14. Formal software specification and program verification	**3**		The tank (reservoir) (**9** slides)
15. Overview of design activities			
16. Structured design	**5**		
17. Object-oriented design	**1**		
18. Implementation			
19. Systematic testing			
20. Functional testing			- Building blocks (**6** slides) - Coverage test (**23** slides)
21. Software metrics		**18**	
22. Maintenance			
23. Reverse engineering		**12**	
24. Quality of software development process and its standardization			
25. Introduction to software ergonomics	**??**		
26. User manuals	**??**		
27. Project management			
28. Configuration and version management			
Assignment 1 – review of requirements specification document	√		
Assignment 2 – cost estimation	√		
Assignment 3 – review of the product model according to structured analysis	√		
Assignment 4 – derive a use case and class diagram for a new software specification			An independent problem (req. spec.)
Assignment 5 – derive a formal specification for a new software subsystem			The queue
Assignment 6 – apply regression testing tool to a new small example program			An independent program (program source)
Assignment 7 – build a classification tree for one use case	√		
Assignment 8 – apply some software metric tools to a new software			An independent program (program source)

2 The main case-study – Seminar Organization

2.1 Topic 5: *Results of the "Analysis and Definition" phase*

The main case study is for the first time mentioned and used in Topic 5: *Results of the "Analysis and Definition" phase*. It is used to show and describe the requirements documents for a software product.

To do:
Develop a preliminary requirements specification *and* requirements specification for a software product that should be:
- of similar size as the current one (based on the number of use cases, for example),
- business-oriented (with data), such that function point method can be applied for cost-estimation,
- of such complexity:
    - that use cases can be used to illustrate include, extend, and generalize relations,
    - that entity-relationship diagrams illustrating important notations can be created,
    - that decision tables illustrating important notations can be created,
    - that data-flow diagrams can be refined reasonably deeply,
    - that class diagram can illustrate all important aspects,
    - etc. (see sections for topics 7 – 13 in this text)

Two mentioned documents should:
- be based on use-cases,
- follow the structure and contents given in [3, 4] (e.g. graphical *and* textual representation of use cases, data, quality expectance, etc.),

Option:
Preliminary requirements specification may *not* be produced. In that case however, one of the assignments should be changed (see *Topic 6: Cost estimation* for further details).

Remark:
It is also possible to use a different document structure (e.g. IEEE standard).

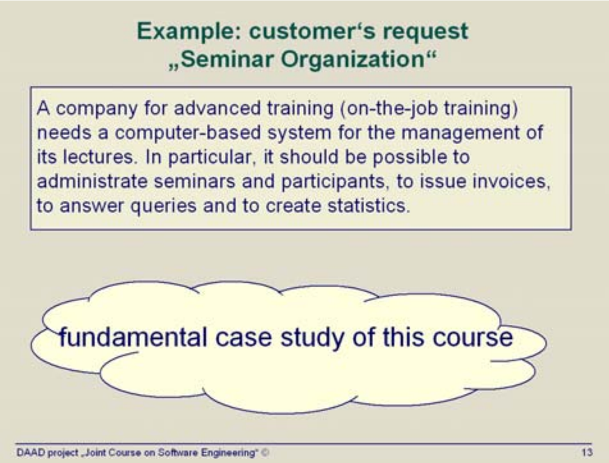
Excerpts from these two documents are used in the lecture (topic 5). Lecture also elaborates on how requirements can change over the time. Therefore,

Option:
Develop a previous version of the requirements. These two documents should follow the structure given in [5, 6] and should be *not* based on use-cases.
Remark:
Since it may be hard to produce these two documents, this activity can be omitted. In that case the topic should be changed by deleting corresponding slides.

Slides where the case study (i.e., the requirements specifications) are used are the following[1]:

[1] Please note that the slide title is a unique identifier of a slide inside the topic.

<u>*General slides*</u> describing only the summary of the product and giving an excerpt from the glossary (part of requirements specification).

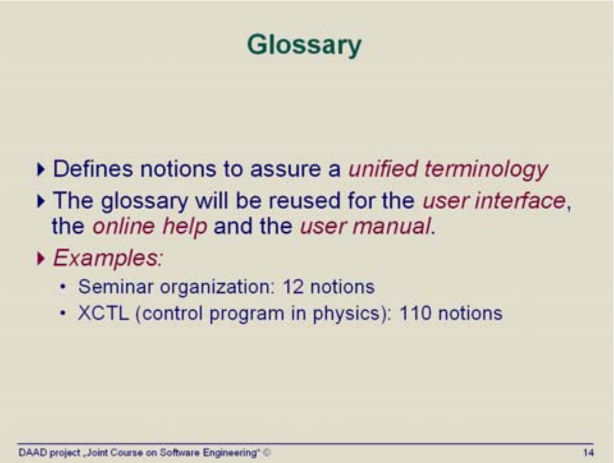


Example: customer‘s request „Seminar Organization“

A company for advanced training (on-the-job training) needs a computer-based system for the management of its lectures. In particular, it should be possible to administrate seminars and participants, to issue invoices, to answer queries and to create statistics.

fundamental case study of this course

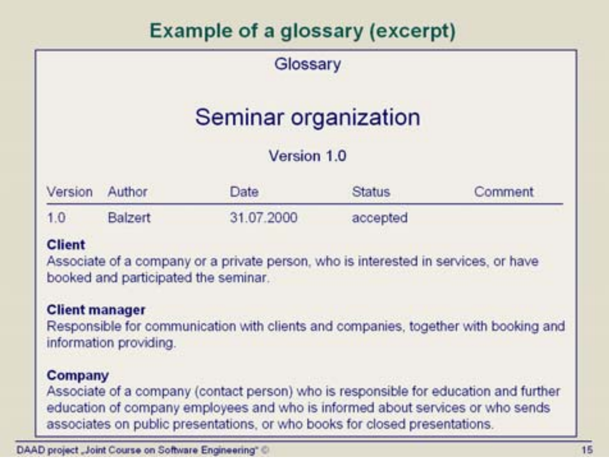
DAAD project „Joint Course on Software Engineering“ © 13



Glossary

- Defines notions to assure a *unified terminology*
- The glossary will be reused for the *user interface*, the *online help* and the *user manual*.
- *Examples:*
  - Seminar organization: 12 notions
  - XCTL (control program in physics): 110 notions

DAAD project „Joint Course on Software Engineering“ © 14



Example of a glossary (excerpt)

Glossary

Seminar organization

Version 1.0

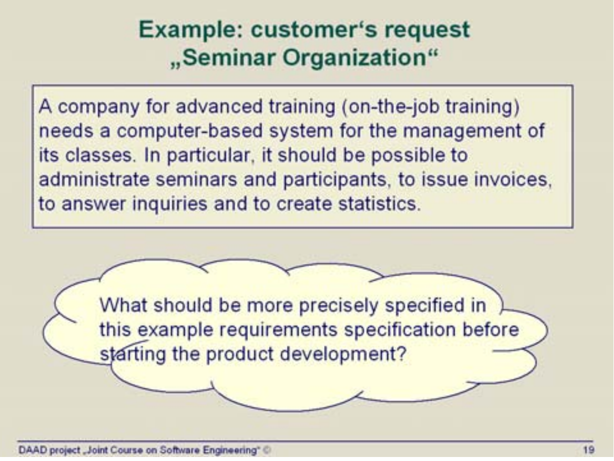
Version	Author	Date	Status	Comment
1.0	Balzert	31.07.2000	accepted	

Client
Associate of a company or a private person, who is interested in services, or have booked and participated the seminar.

Client manager
Responsible for communication with clients and companies, together with booking and information providing.

Company
Associate of a company (contact person) who is responsible for education and further education of company employees and who is informed about services or who sends associates on public presentations, or who books for closed presentations.

DAAD project „Joint Course on Software Engineering“ © 15



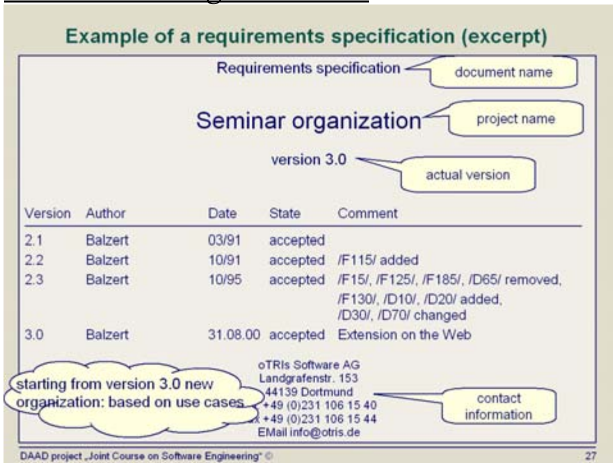
Example: customer‘s request „Seminar Organization“

A company for advanced training (on-the-job training) needs a computer-based system for the management of its classes. In particular, it should be possible to administrate seminars and participants, to issue invoices, to answer inquiries and to create statistics.

What should be more precisely specified in this example requirements specification before starting the product development?

DAAD project „Joint Course on Software Engineering“ © 19

<u>*Slides describing documents*</u> that show characteristic parts of both documents.



Example of a requirements specification (excerpt)

Requirements specification — document name

Seminar organization — project name

version 3.0 — actual version

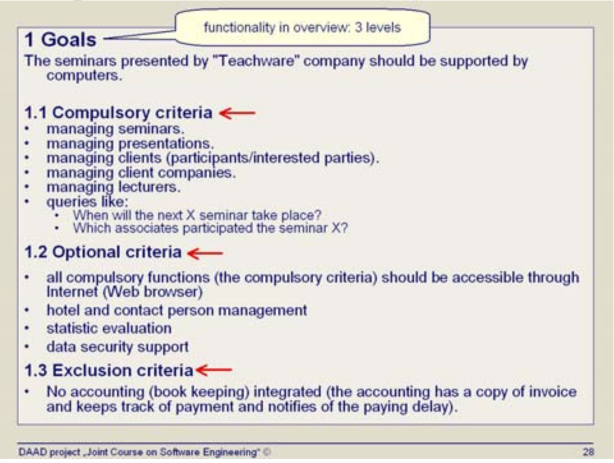
Version	Author	Date	State	Comment
2.1	Balzert	03/91	accepted	
2.2	Balzert	10/91	accepted	/F115/ added
2.3	Balzert	10/95	accepted	/F15/, /F125/, /F185/, /D65/ removed, /F130/, /D10/, /D20/ added, /D30/, /D70/ changed
3.0	Balzert	31.08.00	accepted	Extension on the Web

oTRIs Software AG
Landgrafenstr. 153
44139 Dortmund
+49 (0)231 106 15 40
+49 (0)231 106 15 44
EMail info@otris.de

contact information

starting from version 3.0 new organization: based on use cases

DAAD project „Joint Course on Software Engineering“ © 27



1 Goals

functionality in overview: 3 levels

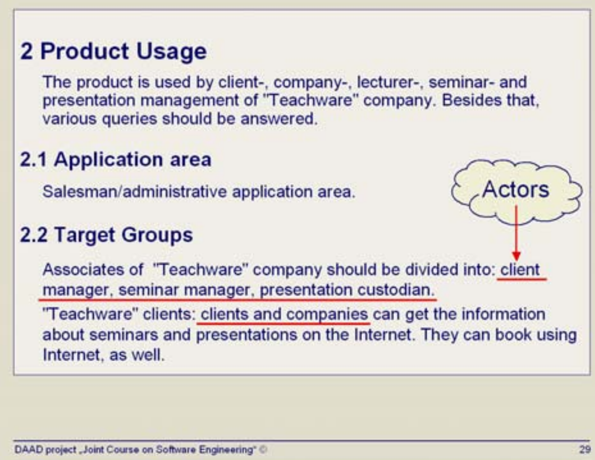
The seminars presented by "Teachware" company should be supported by computers.

1.1 Compulsory criteria ←
- managing seminars.
- managing presentations.
- managing clients (participants/interested parties).
- managing client companies.
- managing lecturers.
- queries like:
  - When will the next X seminar take place?
  - Which associates participated the seminar X?

1.2 Optional criteria ←
- all compulsory functions (the compulsory criteria) should be accessible through Internet (Web browser)
- hotel and contact person management
- statistic evaluation
- data security support

1.3 Exclusion criteria ←
- No accounting (book keeping) integrated (the accounting has a copy of invoice and keeps track of payment and notifies of the paying delay).

DAAD project „Joint Course on Software Engineering“ © 28



2 Product Usage

The product is used by client-, company-, lecturer-, seminar- and presentation management of "Teachware" company. Besides that, various queries should be answered.

2.1 Application area

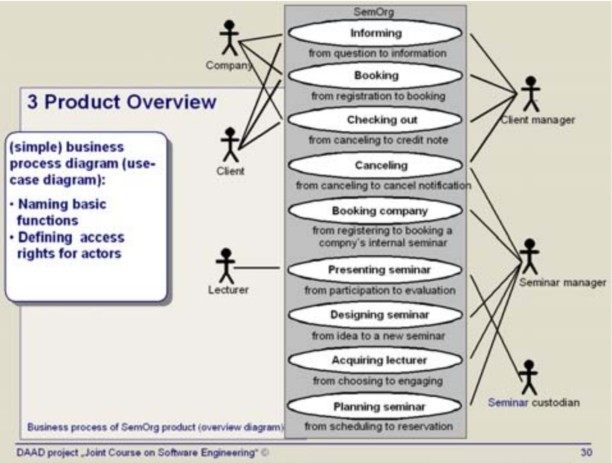
Salesman/administrative application area.

Actors

2.2 Target Groups

Associates of "Teachware" company should be divided into: client manager, seminar manager, presentation custodian.

"Teachware" clients: clients and companies can get the information about seminars and presentations on the Internet. They can book using Internet, as well.



3 Product Overview

(simple) business process diagram (use-case diagram):

- **Naming basic functions**
- **Defining access rights for actors**

SemOrg

Informing from question to information

Booking from registration to booking

Checking out from canceling to credit note

Canceling from canceling to cancel notification

Booking company from registering to booking a compny's internal seminar

Presenting seminar from participation to evaluation

Designing seminar from idea to a new seminar

Acquiring lecturer from choosing to engaging

Planning seminar from scheduling to reservation

Company

Client

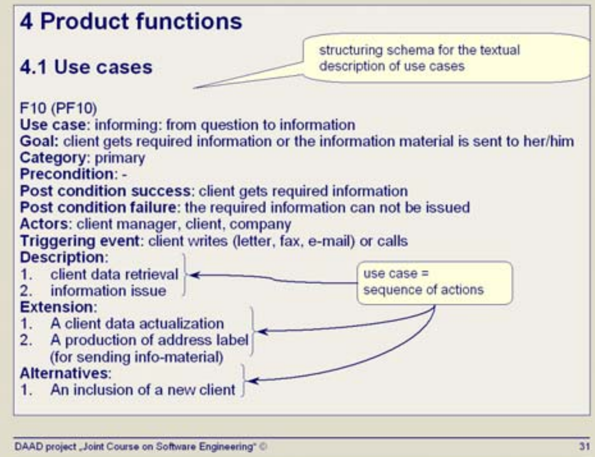
Lecturer

Client manager

Seminar manager

Seminar custodian

Business process of SemOrg product (overview diagram)



4 Product functions

4.1 Use cases

structuring schema for the textual description of use cases

F10 (PF10)

Use case: informing: from question to information
Goal: client gets required information or the information material is sent to her/him
Category: primary
Precondition: -
Post condition success: client gets required information
Post condition failure: the required information can not be issued
Actors: client manager, client, company
Triggering event: client writes (letter, fax, e-mail) or calls
Description:

1. client data retrieval
2. information issue

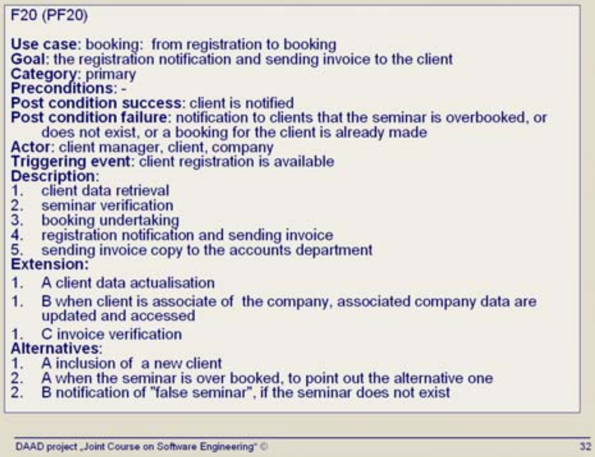
use case = sequence of actions

Extension:

1. A client data actualization
2. A production of address label (for sending info-material)

Alternatives:

1. An inclusion of a new client



F20 (PF20)

Use case: booking: from registration to booking
Goal: the registration notification and sending invoice to the client
Category: primary
Preconditions: -
Post condition success: client is notified
Post condition failure: notification to clients that the seminar is overbooked, or does not exist, or a booking for the client is already made
Actor: client manager, client, company
Triggering event: client registration is available
Description:

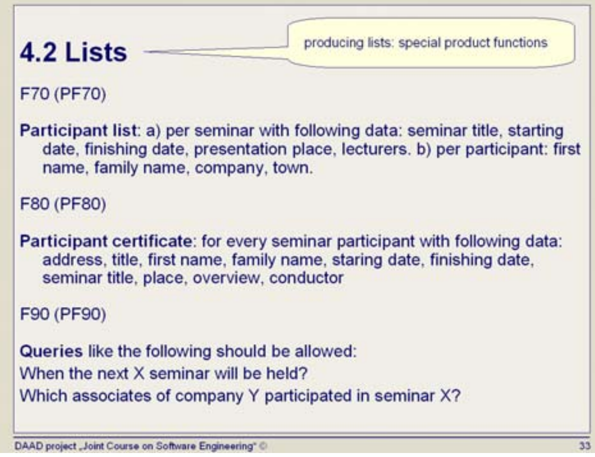
1. client data retrieval
2. seminar verification
3. booking undertaking
4. registration notification and sending invoice
5. sending invoice copy to the accounts department

Extension:

1. A client data actualisation
1. B when client is associate of the company, associated company data are updated and accessed
1. C invoice verification

Alternatives:

1. A inclusion of a new client
2. A when the seminar is over booked, to point out the alternative one
2. B notification of "false seminar", if the seminar does not exist



4.2 Lists

producing lists: special product functions

F70 (PF70)

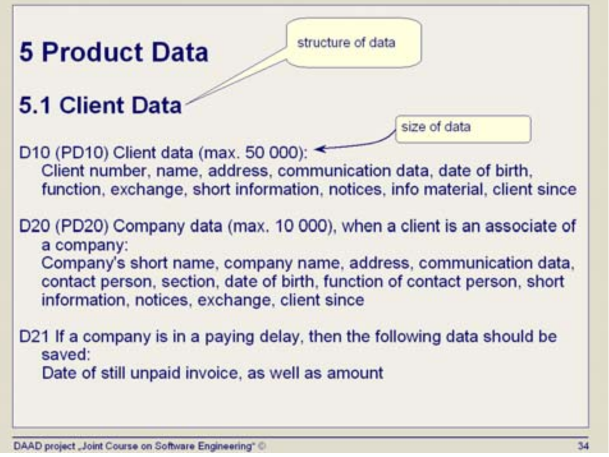
Participant list: a) per seminar with following data: seminar title, starting date, finishing date, presentation place, lecturers. b) per participant: first name, family name, company, town.

F80 (PF80)

Participant certificate: for every seminar participant with following data: address, title, first name, family name, staring date, finishing date, seminar title, place, overview, conductor

F90 (PF90)

Queries like the following should be allowed:
When the next X seminar will be held?
Which associates of company Y participated in seminar X?



5 Product Data

structure of data

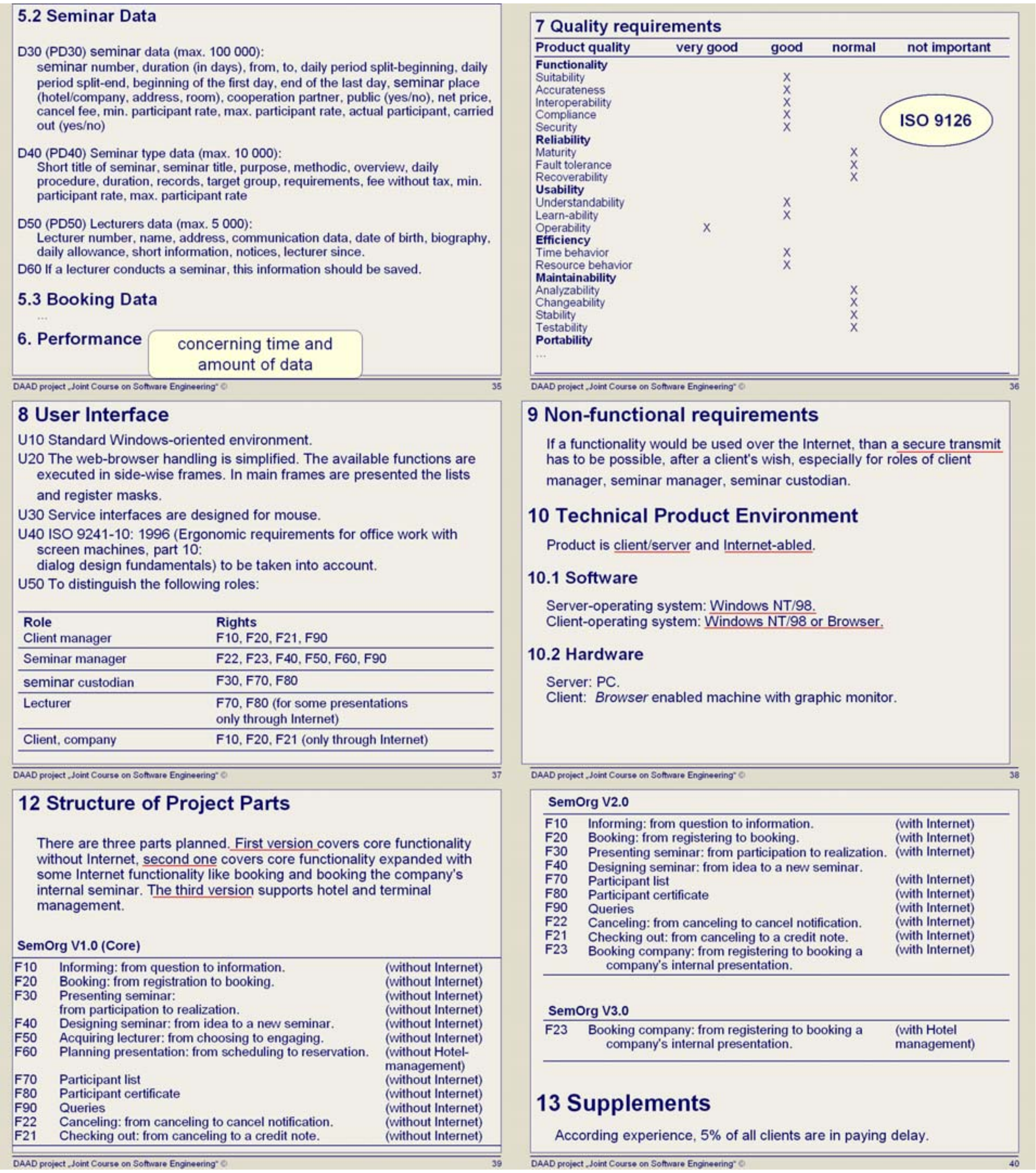
5.1 Client Data

size of data

D10 (PD10) Client data (max. 50 000):
Client number, name, address, communication data, date of birth, function, exchange, short information, notices, info material, client since

D20 (PD20) Company data (max. 10 000), when a client is an associate of a company:
Company's short name, company name, address, communication data, contact person, section, date of birth, function of contact person, short information, notices, exchange, client since

D21 If a company is in a paying delay, then the following data should be saved:
Date of still unpaid invoice, as well as amount



5.2 Seminar Data

D30 (PD30) seminar data (max. 100 000):
seminar number, duration (in days), from, to, daily period split-beginning, daily period split-end, beginning of the first day, end of the last day, seminar place (hotel/company, address, room), cooperation partner, public (yes/no), net price, cancel fee, min. participant rate, max. participant rate, actual participant, carried out (yes/no)

D40 (PD40) Seminar type data (max. 10 000):
Short title of seminar, seminar title, purpose, methodic, overview, daily procedure, duration, records, target group, requirements, fee without tax, min. participant rate, max. participant rate

D50 (PD50) Lecturers data (max. 5 000):
Lecturer number, name, address, communication data, date of birth, biography, daily allowance, short information, notices, lecturer since.

D60 If a lecturer conducts a seminar, this information should be saved.

5.3 Booking Data

...

6. Performance

concerning time and amount of data

DAAD project „Joint Course on Software Engineering" © 35

7 Quality requirements

Product quality	very good	good	normal	not important
Functionality				
Suitability		X		
Accurateness		X		
Interoperability		X		
Compliance		X		
Security		X		
Reliability				
Maturity			X	
Fault tolerance			X	
Recoverability			X	
Usability				
Understandability		X		
Learn-ability		X		
Operability	X			
Efficiency				
Time behavior		X		
Resource behavior		X		
Maintainability				
Analyzability			X	
Changeability			X	
Stability			X	
Testability			X	
Portability				
...				

ISO 9126

DAAD project „Joint Course on Software Engineering" © 36

8 User Interface

U10 Standard Windows-oriented environment.

U20 The web-browser handling is simplified. The available functions are executed in side-wise frames. In main frames are presented the lists and register masks.

U30 Service interfaces are designed for mouse.

U40 ISO 9241-10: 1996 (Ergonomic requirements for office work with screen machines, part 10:
dialog design fundamentals) to be taken into account.

U50 To distinguish the following roles:

Role	Rights
Client manager	F10, F20, F21, F90
Seminar manager	F22, F23, F40, F50, F60, F90
seminar custodian	F30, F70, F80
Lecturer	F70, F80 (for some presentations only through Internet)
Client, company	F10, F20, F21 (only through Internet)

DAAD project „Joint Course on Software Engineering" © 37

9 Non-functional requirements

If a functionality would be used over the Internet, than a secure transmit has to be possible, after a client's wish, especially for roles of client manager, seminar manager, seminar custodian.

10 Technical Product Environment

Product is client/server and Internet-abled.

10.1 Software

Server-operating system: Windows NT/98.
Client-operating system: Windows NT/98 or Browser.

10.2 Hardware

Server: PC.
Client: *Browser* enabled machine with graphic monitor.

DAAD project „Joint Course on Software Engineering" © 38

12 Structure of Project Parts

There are three parts planned. First version covers core functionality without Internet, second one covers core functionality expanded with some Internet functionality like booking and booking the company's internal seminar. The third version supports hotel and terminal management.

SemOrg V1.0 (Core)

F10	Informing: from question to information.	(without Internet)
F20	Booking: from registration to booking.	(without Internet)
F30	Presenting seminar: from participation to realization.	(without Internet) (without Internet)
F40	Designing seminar: from idea to a new seminar.	(without Internet)
F50	Acquiring lecturer: from choosing to engaging.	(without Internet)
F60	Planning presentation: from scheduling to reservation.	(without Hotel-management)
F70	Participant list	(without Internet)
F80	Participant certificate	(without Internet)
F90	Queries	(without Internet)
F22	Canceling: from canceling to cancel notification.	(without Internet)
F21	Checking out: from canceling to a credit note.	(without Internet)

DAAD project „Joint Course on Software Engineering" © 39

SemOrg V2.0

F10	Informing: from question to information.	(with Internet)
F20	Booking: from registering to booking.	(with Internet)
F30	Presenting seminar: from participation to realization.	(with Internet)
F40	Designing seminar: from idea to a new seminar.	
F70	Participant list	(with Internet)
F80	Participant certificate	(with Internet)
F90	Queries	(with Internet)
F22	Canceling: from canceling to cancel notification.	(with Internet)
F21	Checking out: from canceling to a credit note.	(with Internet)
F23	Booking company: from registering to booking a company's internal presentation.	(with Internet)

SemOrg V3.0

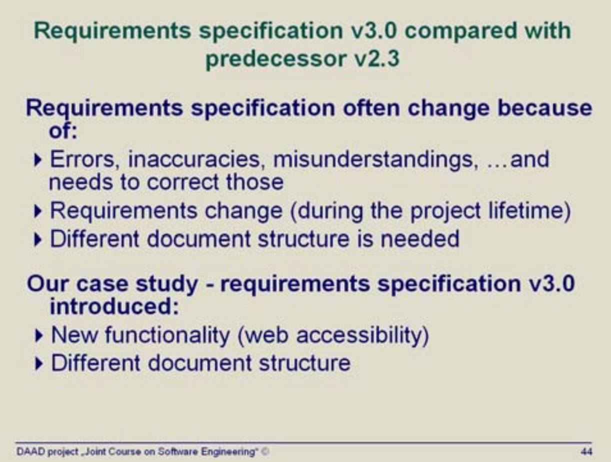
F23	Booking company: from registering to booking a company's internal presentation.	(with Hotel management)

13 Supplements

According experience, 5% of all clients are in paying delay.

DAAD project „Joint Course on Software Engineering" © 40

Slides comparing two versions of requirements ***(optional)*** that summarize differences between the previous and the current requirement specifications.



Requirements specification v3.0 compared with predecessor v2.3

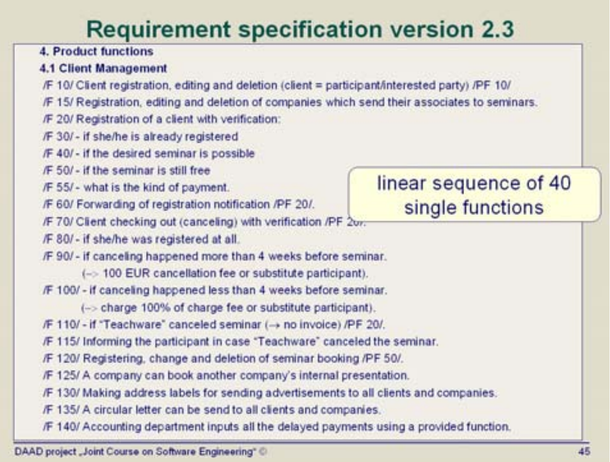
Requirements specification often change because of:

- Errors, inaccuracies, misunderstandings, …and needs to correct those
- Requirements change (during the project lifetime)
- Different document structure is needed

Our case study - requirements specification v3.0 introduced:

- New functionality (web accessibility)
- Different document structure

DAAD project „Joint Course on Software Engineering" © 44



Requirement specification version 2.3

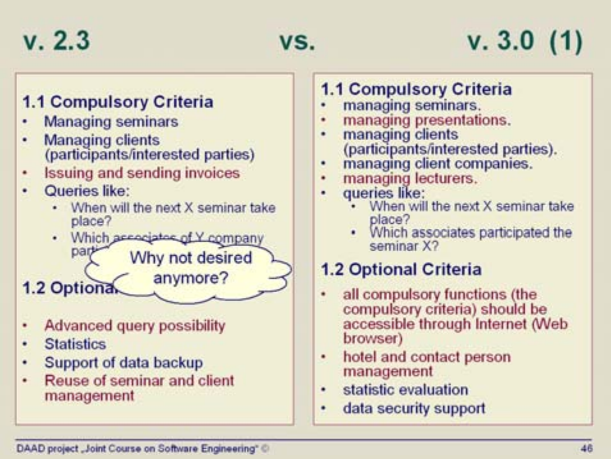
4. Product functions

4.1 Client Management

/F 10/ Client registration, editing and deletion (client = participant/interested party) /PF 10/
/F 15/ Registration, editing and deletion of companies which send their associates to seminars.
/F 20/ Registration of a client with verification:
/F 30/ - if she/he is already registered
/F 40/ - if the desired seminar is possible
/F 50/ - if the seminar is still free
/F 55/ - what is the kind of payment.
/F 60/ Forwarding of registration notification /PF 20/.
/F 70/ Client checking out (canceling) with verification /PF 20/.
/F 80/ - if she/he was registered at all.
/F 90/ - if canceling happened more than 4 weeks before seminar.
(--> 100 EUR cancellation fee or substitute participant).
/F 100/ - if canceling happened less than 4 weeks before seminar.
(--> charge 100% of charge fee or substitute participant).
/F 110/ - if "Teachware" canceled seminar (→ no invoice) /PF 20/.
/F 115/ Informing the participant in case "Teachware" canceled the seminar.
/F 120/ Registering, change and deletion of seminar booking /PF 50/.
/F 125/ A company can book another company's internal presentation.
/F 130/ Making address labels for sending advertisements to all clients and companies.
/F 135/ A circular letter can be send to all clients and companies.
/F 140/ Accounting department inputs all the delayed payments using a provided function.

linear sequence of 40 single functions

DAAD project „Joint Course on Software Engineering" © 45



v. 2.3 vs. v. 3.0 (1)

1.1 Compulsory Criteria

- Managing seminars
- Managing clients (participants/interested parties)
- Issuing and sending invoices
- Queries like:
  - When will the next X seminar take place?
  - Which associates of Y company part

1.2 Optiona

- Advanced query possibility
- Statistics
- Support of data backup
- Reuse of seminar and client management

Why not desired anymore?

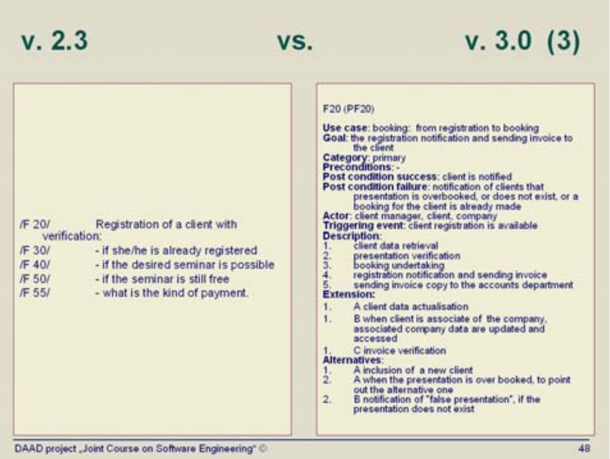
1.1 Compulsory Criteria

- managing seminars.
- managing presentations.
- managing clients (participants/interested parties).
- managing client companies.
- managing lecturers.
- queries like:
  - When will the next X seminar take place?
  - Which associates participated the seminar X?

1.2 Optional Criteria

- all compulsory functions (the compulsory criteria) should be accessible through Internet (Web browser)
- hotel and contact person management
- statistic evaluation
- data security support

DAAD project „Joint Course on Software Engineering" © 46



v. 2.3 vs. v. 3.0 (3)

/F 20/ Registration of a client with verification:
/F 30/ - if she/he is already registered
/F 40/ - if the desired seminar is possible
/F 50/ - if the seminar is still free
/F 55/ - what is the kind of payment.

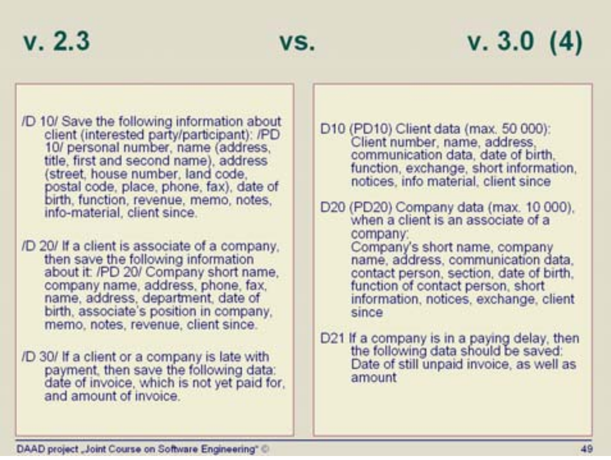
F20 (PF20)

Use case: booking: from registration to booking
Goal: the registration notification and sending invoice to the client
Category: primary
Preconditions: -
Post condition success: client is notified
Post condition failure: notification of clients that presentation is overbooked, or does not exist, or a booking for the client is already made
Actor: client manager, client, company
Triggering event: client registration is available
Description:
1. client data retrieval
2. presentation verification
3. booking undertaking
4. registration notification and sending invoice
5. sending invoice copy to the accounts department

Extension:
1. A client data actualisation
1. B when client is associate of the company, associated company data are updated and accessed
1. C invoice verification

Alternatives:
1. A inclusion of a new client
2. A when the presentation is over booked, to point out the alternative one
2. B notification of "false presentation", if the presentation does not exist

DAAD project „Joint Course on Software Engineering" © 48



v. 2.3 vs. v. 3.0 (4)

/D 10/ Save the following information about client (interested party/participant): /PD 10/ personal number, name (address, title, first and second name), address (street, house number, land code, postal code, place, phone, fax), date of birth, function, revenue, memo, notes, info-material, client since.

/D 20/ If a client is associate of a company, then save the following information about it: /PD 20/ Company short name, company name, address, phone, fax, name, address, department, date of birth, associate's position in company, memo, notes, revenue, client since.

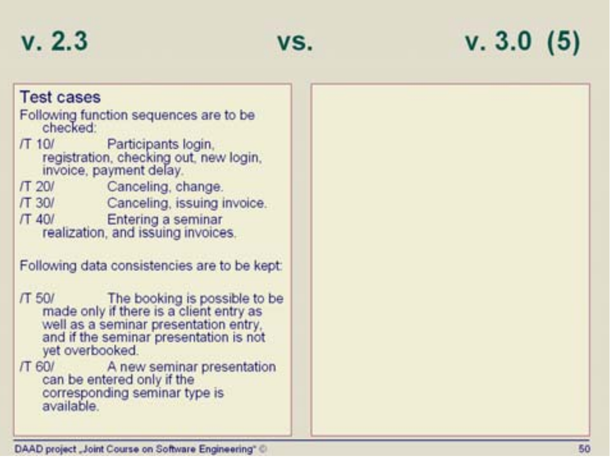
/D 30/ If a client or a company is late with payment, then save the following data: date of invoice, which is not yet paid for, and amount of invoice.

D10 (PD10) Client data (max. 50 000): Client number, name, address, communication data, date of birth, function, exchange, short information, notices, info material, client since

D20 (PD20) Company data (max. 10 000), when a client is an associate of a company:
Company's short name, company name, address, communication data, contact person, section, date of birth, function of contact person, short information, notices, exchange, client since

D21 If a company is in a paying delay, then the following data should be saved: Date of still unpaid invoice, as well as amount

DAAD project „Joint Course on Software Engineering" © 49



v. 2.3 vs. v. 3.0 (5)

Test cases

Following function sequences are to be checked:
/T 10/ Participants login, registration, checking out, new login, invoice, payment delay.
/T 20/ Canceling, change.
/T 30/ Canceling, issuing invoice.
/T 40/ Entering a seminar realization, and issuing invoices.

Following data consistencies are to be kept:

/T 50/ The booking is possible to be made only if there is a client entry as well as a seminar presentation entry, and if the seminar presentation is not yet overbooked.
/T 60/ A new seminar presentation can be entered only if the corresponding seminar type is available.

DAAD project „Joint Course on Software Engineering" © 50

2.2 Topic 6: *Cost Estimation*

Documents produced in the previous step are used to calculate cost estimation that will be partly shown during this topic.

To do:
Develop a cost estimation calculation using a function point method, based on preliminary requirements specification produced in previous step.
Option:

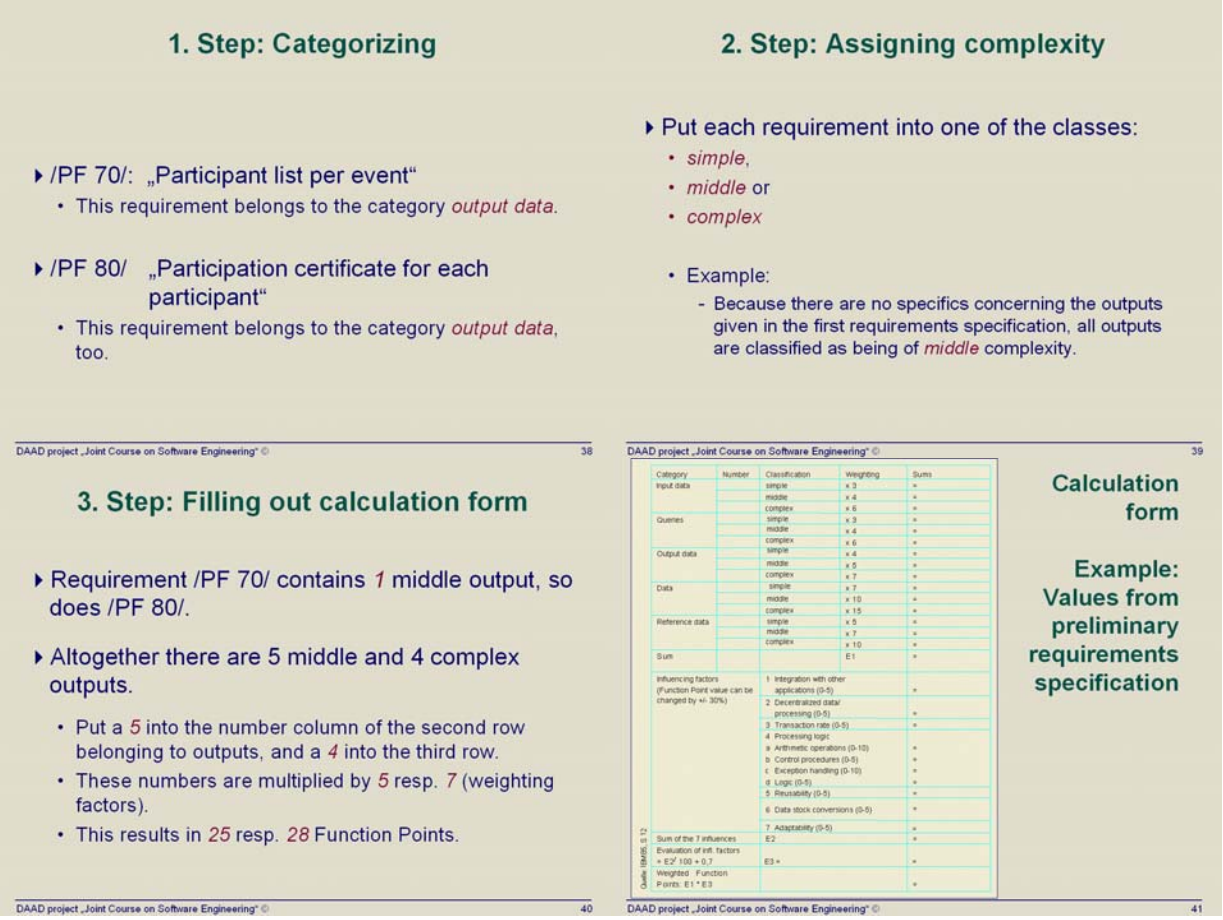
If the preliminary requirements specification *has not* been produced in the previous step, then cost estimation must be shown on requirements specification! Since this was intended as a student assignment (see later), in this case another student assignment (i.e. example) must be devised.

Remark:

It is also possible to use another cost estimation method (e.g. COCOMO) but it should be done only as an additional method. Function point (at this time) has the priority.

Slides where the cost estimation calculation is used are the following[2]:

Introductory slides used to support the introduction of FP method.



1. Step: Categorizing

- /PF 70/: „Participant list per event"
  - This requirement belongs to the category *output data*.
- /PF 80/ „Participation certificate for each participant"
  - This requirement belongs to the category *output data*, too.

DAAD project „Joint Course on Software Engineering" © 38

2. Step: Assigning complexity

- Put each requirement into one of the classes:
  - *simple*,
  - *middle* or
  - *complex*
  - Example:
    - Because there are no specifics concerning the outputs given in the first requirements specification, all outputs are classified as being of *middle* complexity.

DAAD project „Joint Course on Software Engineering" © 39

3. Step: Filling out calculation form

- Requirement /PF 70/ contains *1* middle output, so does /PF 80/.
- Altogether there are 5 middle and 4 complex outputs.
  - Put a *5* into the number column of the second row belonging to outputs, and a *4* into the third row.
  - These numbers are multiplied by *5* resp. *7* (weighting factors).
  - This results in *25* resp. *28* Function Points.

DAAD project „Joint Course on Software Engineering" © 40

Category	Number	Classification	Weighting	Sums
Input data		simple	x 3	=
		middle	x 4	=
		complex	x 6	=
Queries		simple	x 3	=
		middle	x 4	=
		complex	x 6	=
Output data		simple	x 4	=
		middle	x 5	=
		complex	x 7	=
Data		simple	x 7	=
		middle	x 10	=
		complex	x 15	=
Reference data		simple	x 5	=
		middle	x 7	=
		complex	x 10	=
Sum			E1	=
Influencing factors (Function Point value can be changed by +/- 30%)		1 Integration with other applications (0-5)		=
		2 Decentralized data/ processing (0-5)		=
		3 Transaction rate (0-5)		=
		4 Processing logic		
		a Arithmetic operations (0-10)		=
		b Control procedures (0-5)		=
		c Exception handling (0-10)		=
		d Logic (0-5)		=
		5 Reusability (0-5)		=
		6 Data stock conversions (0-5)		=
		7 Adaptability (0-5)		=
Sum of the 7 influences		E2		=
Evaluation of infl. factors = E2/ 100 + 0,7		E3 =		=
Weighted Function Points: E1 * E3				=

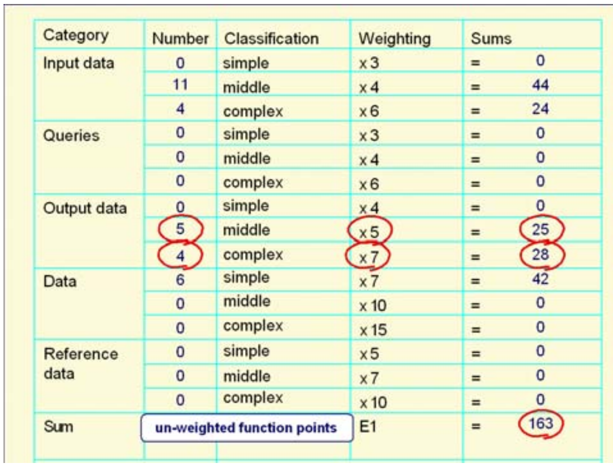
Quelle: IBM85, S.12

Calculation form

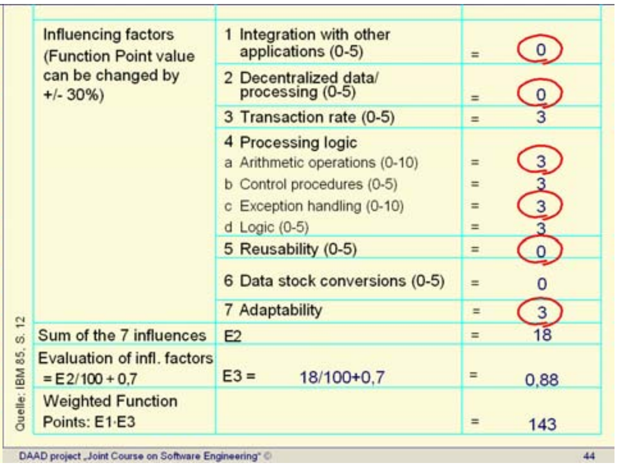
Example: Values from preliminary requirements specification

DAAD project „Joint Course on Software Engineering" © 41

[2] Please note that the slide title is a unique identifier of a slide inside the topic.



Category	Number	Classification	Weighting	Sums	
Input data	0	simple	x 3	=	0
	11	middle	x 4	=	44
	4	complex	x 6	=	24
Queries	0	simple	x 3	=	0
	0	middle	x 4	=	0
	0	complex	x 6	=	0
Output data	0	simple	x 4	=	0
	5	middle	x 5	=	25
	4	complex	x 7	=	28
Data	6	simple	x 7	=	42
	0	middle	x 10	=	0
	0	complex	x 15	=	0
Reference data	0	simple	x 5	=	0
	0	middle	x 7	=	0
	0	complex	x 10	=	0
Sum	un-weighted function points		E1	=	163

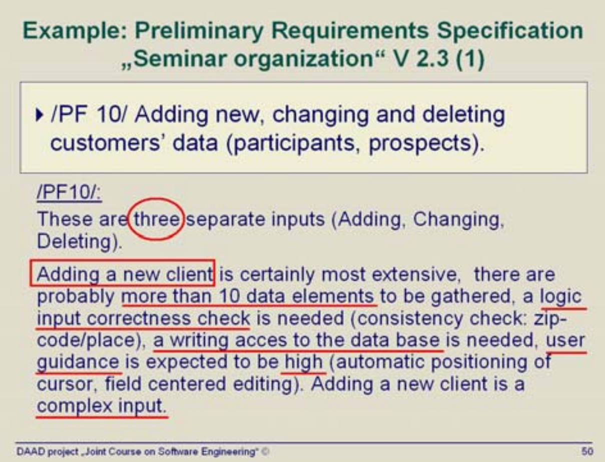


Influencing factors (Function Point value can be changed by +/- 30%)	1 Integration with other applications (0-5)	=	0
	2 Decentralized data/ processing (0-5)	=	0
	3 Transaction rate (0-5)	=	3
	4 Processing logic		
	a Arithmetic operations (0-10)	=	3
	b Control procedures (0-5)	=	3
	c Exception handling (0-10)	=	3
	d Logic (0-5)	=	3
	5 Reusability (0-5)	=	0
	6 Data stock conversions (0-5)	=	0
	7 Adaptability	=	3
Sum of the 7 influences	E2	=	18
Evaluation of infl. factors = E2/100 + 0,7	E3 = 18/100+0,7	=	0,88
Weighted Function Points: E1·E3		=	143

Quelle: IBM 85, S. 12

DAAD project „Joint Course on Software Engineering“ © 44

Detailed slides showing detailed parts of calculation.



Example: Preliminary Requirements Specification „Seminar organization“ V 2.3 (1)

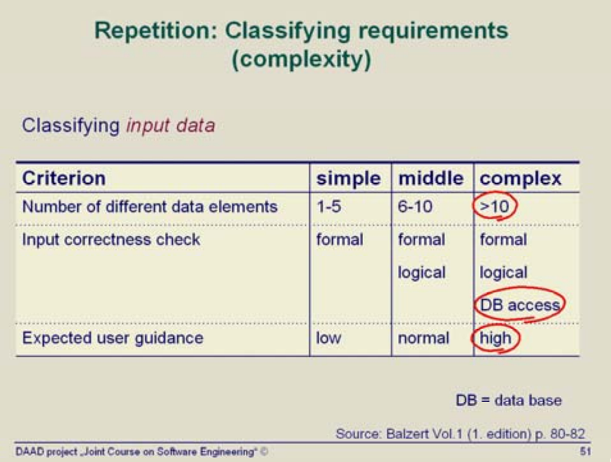
- /PF 10/ Adding new, changing and deleting customers' data (participants, prospects).

/PF10/:

These are three separate inputs (Adding, Changing, Deleting).

Adding a new client is certainly most extensive, there are probably more than 10 data elements to be gathered, a logic input correctness check is needed (consistency check: zip-code/place), a writing acces to the data base is needed, user guidance is expected to be high (automatic positioning of cursor, field centered editing). Adding a new client is a complex input.

DAAD project „Joint Course on Software Engineering“ © 50



Repetition: Classifying requirements (complexity)

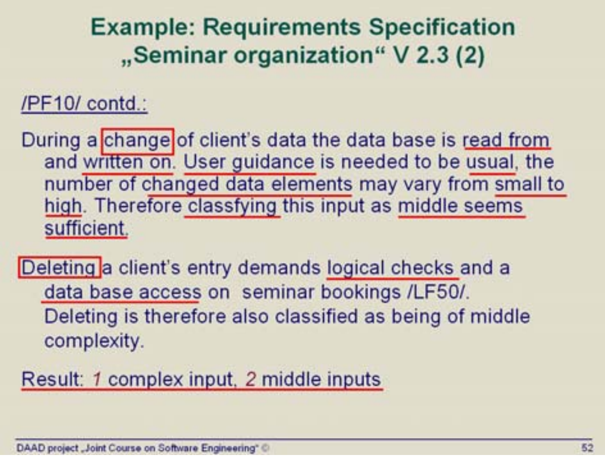
Classifying *input data*

Criterion	simple	middle	complex
Number of different data elements	1-5	6-10	>10
Input correctness check	formal	formal logical	formal logical DB access
Expected user guidance	low	normal	high

DB = data base

Source: Balzert Vol.1 (1. edition) p. 80-82

DAAD project „Joint Course on Software Engineering“ © 51



Example: Requirements Specification „Seminar organization“ V 2.3 (2)

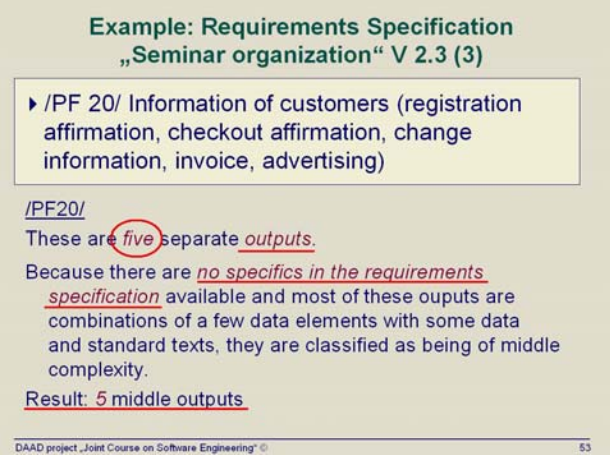
/PF10/ contd.:

During a change of client's data the data base is read from and written on. User guidance is needed to be usual, the number of changed data elements may vary from small to high. Therefore classfying this input as middle seems sufficient.

Deleting a client's entry demands logical checks and a data base access on seminar bookings /LF50/. Deleting is therefore also classified as being of middle complexity.

Result: *1* complex input, *2* middle inputs

DAAD project „Joint Course on Software Engineering“ © 52



Example: Requirements Specification „Seminar organization“ V 2.3 (3)

- /PF 20/ Information of customers (registration affirmation, checkout affirmation, change information, invoice, advertising)

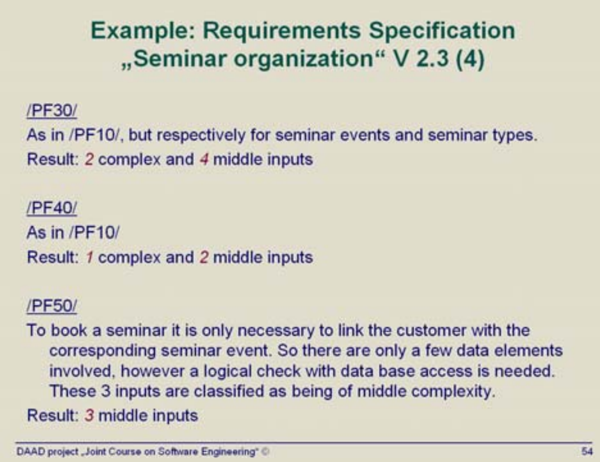
/PF20/

These are *five* separate *outputs.*

Because there are *no specifics in the requirements specification* available and most of these ouputs are combinations of a few data elements with some data and standard texts, they are classified as being of middle complexity.

Result: *5* middle outputs

DAAD project „Joint Course on Software Engineering“ © 53

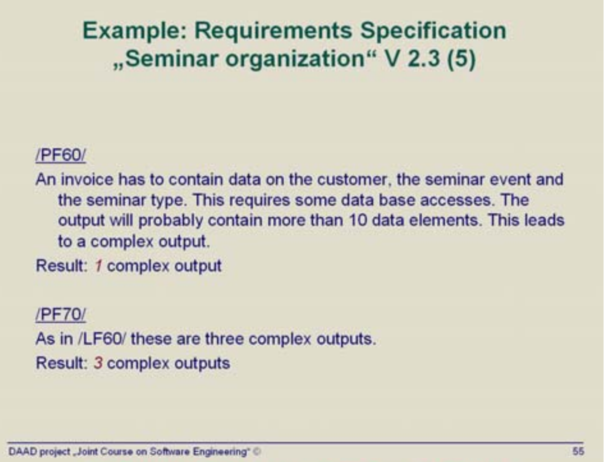


Example: Requirements Specification „Seminar organization“ V 2.3 (4)

/PF30/
As in /PF10/, but respectively for seminar events and seminar types.
Result: *2* complex and *4* middle inputs

/PF40/
As in /PF10/
Result: *1* complex and *2* middle inputs

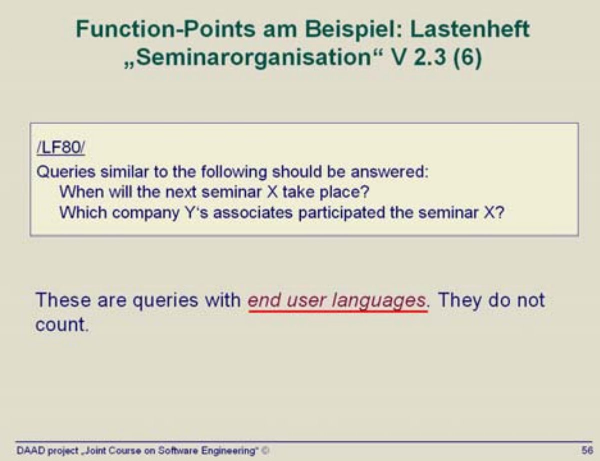
/PF50/
To book a seminar it is only necessary to link the customer with the corresponding seminar event. So there are only a few data elements involved, however a logical check with data base access is needed. These 3 inputs are classified as being of middle complexity.
Result: *3* middle inputs



Example: Requirements Specification „Seminar organization“ V 2.3 (5)

/PF60/
An invoice has to contain data on the customer, the seminar event and the seminar type. This requires some data base accesses. The output will probably contain more than 10 data elements. This leads to a complex output.
Result: *1* complex output

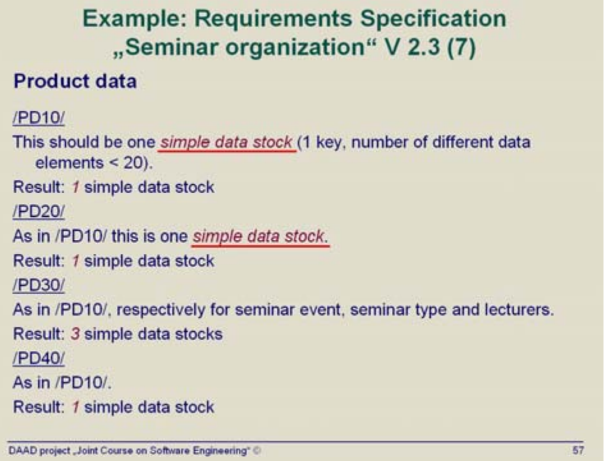
/PF70/
As in /LF60/ these are three complex outputs.
Result: *3* complex outputs



Function-Points am Beispiel: Lastenheft „Seminarorganisation“ V 2.3 (6)

/LF80/
Queries similar to the following should be answered:
When will the next seminar X take place?
Which company Y‘s associates participated the seminar X?

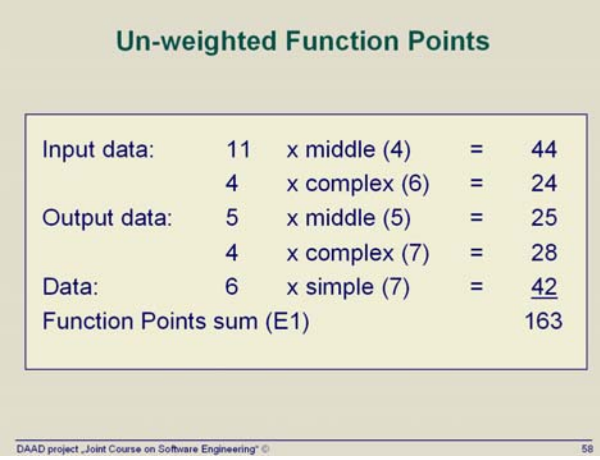
These are queries with *end user languages*. They do not count.



Example: Requirements Specification „Seminar organization“ V 2.3 (7)

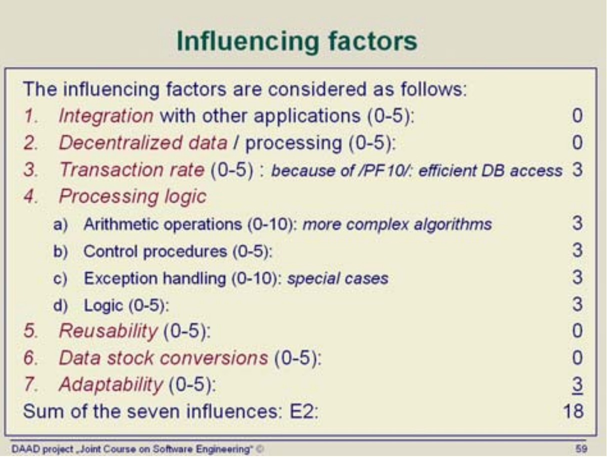
Product data

/PD10/
This should be one *simple data stock* (1 key, number of different data elements < 20).
Result: *1* simple data stock
/PD20/
As in /PD10/ this is one *simple data stock.*
Result: *1* simple data stock
/PD30/
As in /PD10/, respectively for seminar event, seminar type and lecturers.
Result: *3* simple data stocks
/PD40/
As in /PD10/.
Result: *1* simple data stock



Un-weighted Function Points

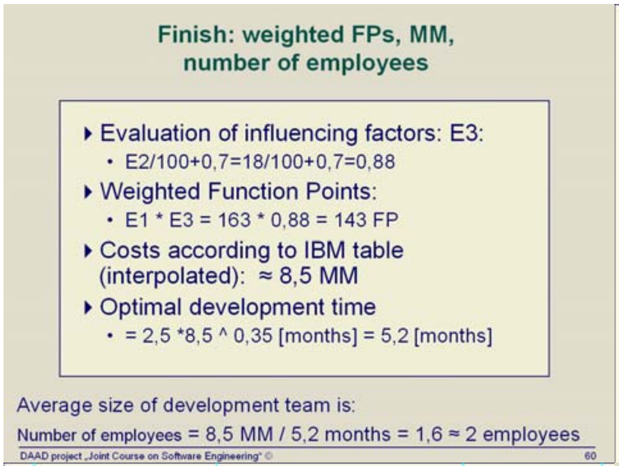
Input data:	11	x middle (4)	=	44
	4	x complex (6)	=	24
Output data:	5	x middle (5)	=	25
	4	x complex (7)	=	28
Data:	6	x simple (7)	=	42
Function Points sum (E1)				163



Influencing factors

The influencing factors are considered as follows:

1. *Integration* with other applications (0-5):	0
2. *Decentralized data* / processing (0-5):	0
3. *Transaction rate* (0-5) : *because of /PF10/: efficient DB access*	3
4. *Processing logic*	
a) Arithmetic operations (0-10): *more complex algorithms*	3
b) Control procedures (0-5):	3
c) Exception handling (0-10): *special cases*	3
d) Logic (0-5):	3
5. *Reusability* (0-5):	0
6. *Data stock conversions* (0-5):	0
7. *Adaptability* (0-5):	3
Sum of the seven influences: E2:	18



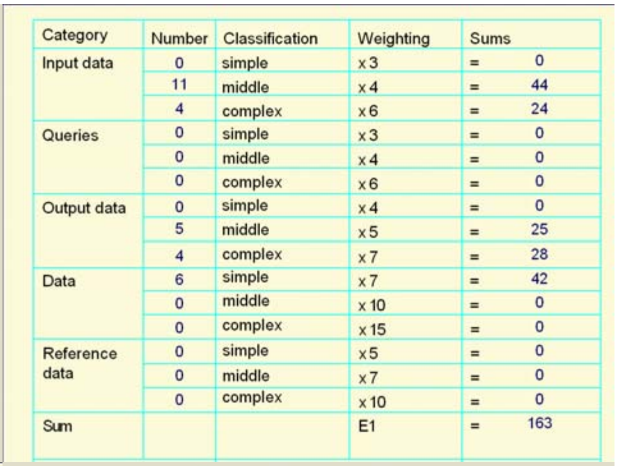
Finish: weighted FPs, MM, number of employees

- Evaluation of influencing factors: E3:
  - E2/100+0,7=18/100+0,7=0,88
- Weighted Function Points:
  - E1 * E3 = 163 * 0,88 = 143 FP
- Costs according to IBM table (interpolated): ≈ 8,5 MM
- Optimal development time
  - = 2,5 *8,5 ^ 0,35 [months] = 5,2 [months]

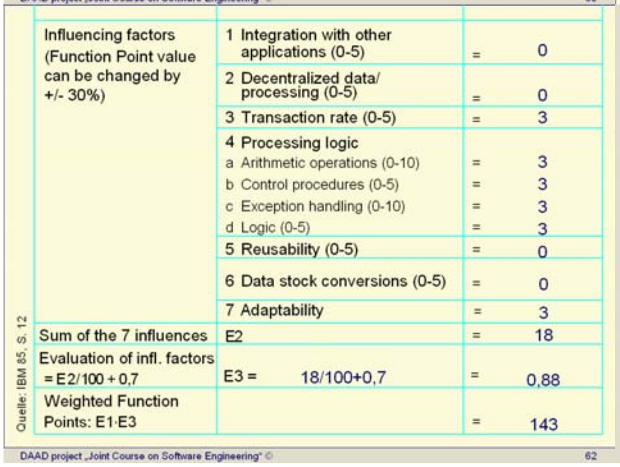
Average size of development team is:

Number of employees = 8,5 MM / 5,2 months = 1,6 ≈ 2 employees

DAAD project „Joint Course on Software Engineering" © 60



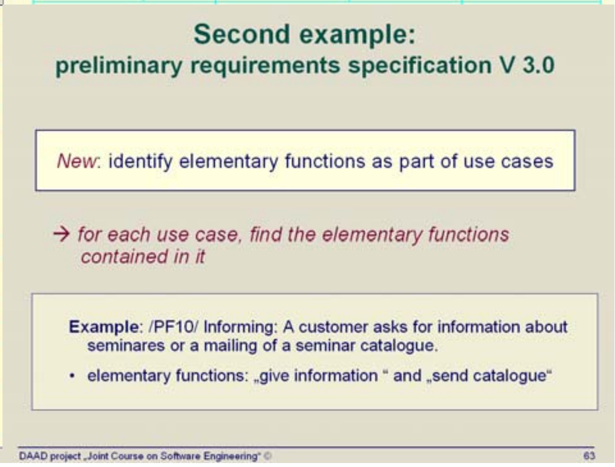
Category	Number	Classification	Weighting	Sums	
Input data	0	simple	x 3	=	0
	11	middle	x 4	=	44
	4	complex	x 6	=	24
Queries	0	simple	x 3	=	0
	0	middle	x 4	=	0
	0	complex	x 6	=	0
Output data	0	simple	x 4	=	0
	5	middle	x 5	=	25
	4	complex	x 7	=	28
Data	6	simple	x 7	=	42
	0	middle	x 10	=	0
	0	complex	x 15	=	0
Reference data	0	simple	x 5	=	0
	0	middle	x 7	=	0
	0	complex	x 10	=	0
Sum			E1	=	163



Influencing factors (Function Point value can be changed by +/- 30%)	1 Integration with other applications (0-5)	=	0
	2 Decentralized data/ processing (0-5)	=	0
	3 Transaction rate (0-5)	=	3
	4 Processing logic		
	a Arithmetic operations (0-10)	=	3
	b Control procedures (0-5)	=	3
	c Exception handling (0-10)	=	3
	d Logic (0-5)	=	3
	5 Reusability (0-5)	=	0
	6 Data stock conversions (0-5)	=	0
	7 Adaptability	=	3
Sum of the 7 influences	E2	=	18
Evaluation of infl. factors = E2/100 + 0,7	E3 = 18/100+0,7	=	0,88
Weighted Function Points: E1·E3		=	143

Quelle: IBM 85, S. 12

DAAD project „Joint Course on Software Engineering" © 62



Second example: preliminary requirements specification V 3.0

New: identify elementary functions as part of use cases

→ *for each use case, find the elementary functions contained in it*

Example: /PF10/ Informing: A customer asks for information about seminares or a mailing of a seminar catalogue.

- elementary functions: „give information " and „send catalogue"

DAAD project „Joint Course on Software Engineering" © 63

2.3 Topic 7: *Basic concepts of the functional view*

Requirements for a software product should now be analyzed according to several methodologies/views. First we take into consideration a functional view and should illustrate function tree, data flow diagram, and use case diagram on the requirements specification of the case study.

To do:

- Develop a *full* data-flow diagram of requirements – it will be needed also later.
- Develop a function tree of main functions of requirements (function tree is implicitly contained in a data-flow diagram, so it just have to be recognized).

Option:

In fact not the full data flow diagram is needed, but at least three subsystems have to be fully developed.

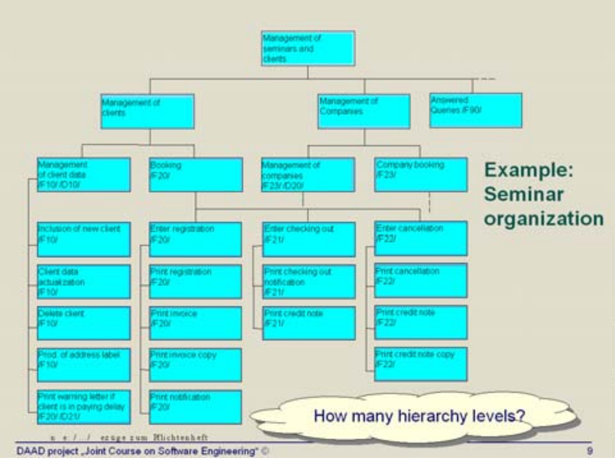
Remark:

If requirements specifications are developed as requested previously, then there is no special activity related to use cases in this lecture – we shall just use some excerpts from the already produced document.

This topic also elaborate on the difference between functions and use cases near the end, using the example from two versions of requirement specification: the old one without use cases, and the new one with use cases.

Remark:
If the previous version of requirements has not been produced earlier, then this elaboration must be illustrated differently in this topic, i.e., changes will be necessary.

Slides on function tree.



Management of seminars and clients

Management of clients

Management of Companies

Answered Queries /F90/

Management of client data /F10/ /D10/

Booking /F20/

Management of companies /F23/ /D20/

Company booking /F23/

Example: Seminar organization

Inclusion of new client /F10/

Client data actualization /F10/

Delete client /F10/

Prod. of address label /F10/

Print warning letter if client is in paying delay /F20/ /D21/

Enter registration /F20/

Print registration /F20/

Print invoice /F20/

Print invoice copy /F20/

Print notification /F20/

Enter checking out /F21/

Print checking out notification /F21/

Print credit note /F21/

Enter cancellation /F22/

Print cancellation /F22/

Print credit note /F22/

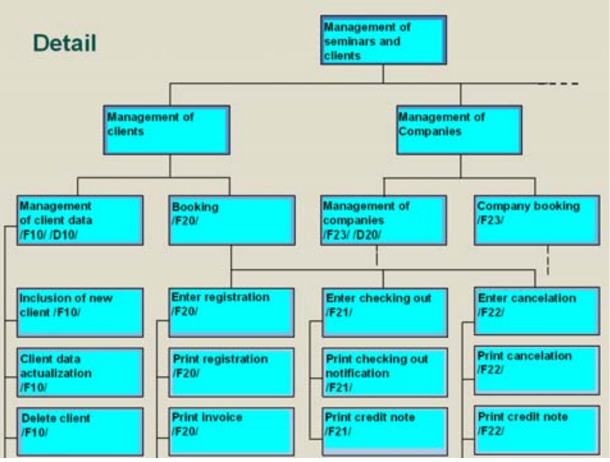
Print credit note copy /F22/

How many hierarchy levels?

n e /.../ ezüge zum Pflichtenheft

DAAD project „Joint Course on Software Engineering" ©

9



Detail

Management of seminars and clients

Management of clients

Management of Companies

Management of client data /F10/ /D10/

Booking /F20/

Management of companies /F23/ /D20/

Company booking /F23/

Inclusion of new client /F10/

Client data actualization /F10/

Delete client /F10/

Enter registration /F20/

Print registration /F20/

Print invoice /F20/

Enter checking out /F21/

Print checking out notification /F21/

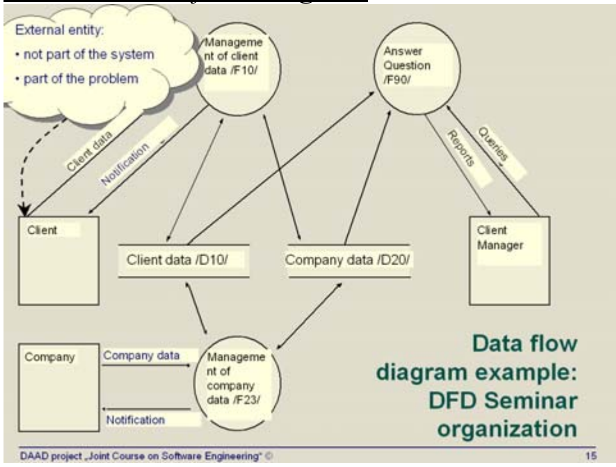
Print credit note /F21/

Enter cancelation /F22/

Print cancelation /F22/

Print credit note /F22/

Slides on data-flow diagram.



External entity:
- not part of the system
- part of the problem

Manageme nt of client data /F10/

Answer Question /F90/

Client data

Notification

Reports

Queries

Client

Client data /D10/

Company data /D20/

Client Manager

Company

Company data

Notification

Manageme nt of company data /F23/

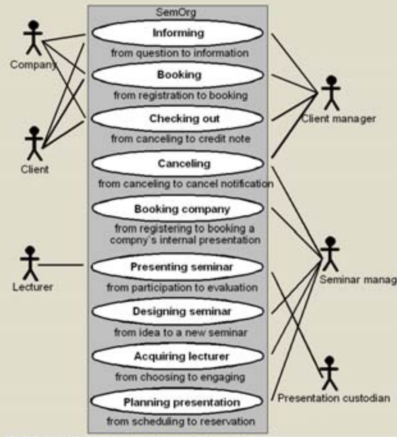
Data flow diagram example: DFD Seminar organization

DAAD project „Joint Course on Software Engineering" ©

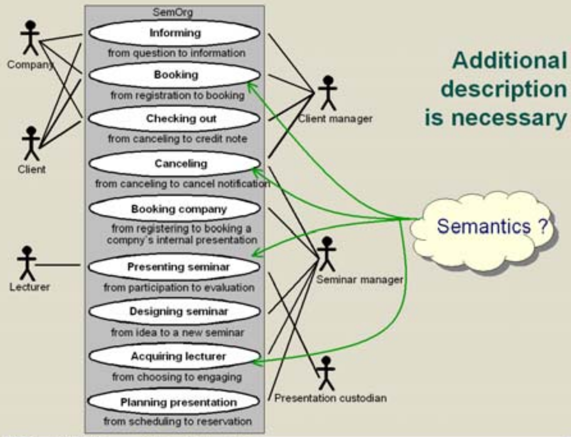
15

Slides on use cases.

Example: Seminar organization


SemOrg
Informing
from question to information
Booking
from registration to booking
Checking out
from canceling to credit note
Canceling
from canceling to cancel notification
Booking company
from registering to booking a compny´s internal presentation
Presenting seminar
from participation to evaluation
Designing seminar
from idea to a new seminar
Acquiring lecturer
from choosing to engaging
Planning presentation
from scheduling to reservation
Company
Client
Lecturer
Client manager
Seminar manager
Presentation custodian

Additional description is necessary


SemOrg
Informing
from question to information
Booking
from registration to booking
Checking out
from canceling to credit note
Canceling
from canceling to cancel notification
Booking company
from registering to booking a compny´s internal presentation
Presenting seminar
from participation to evaluation
Designing seminar
from idea to a new seminar
Acquiring lecturer
from choosing to engaging
Planning presentation
from scheduling to reservation
Company
Client
Lecturer
Client manager
Seminar manager
Presentation custodian
Semantics ?

Description of use cases: example ‚Informing‘
(from: requirements specification „seminar organization“)

structuring schema for the textual description of use cases

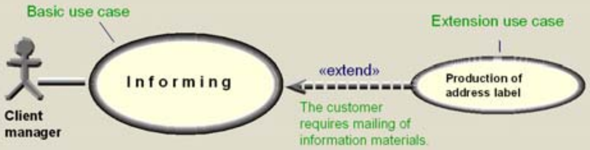
F10 (PF10)
Use case: informing: from question to information
Goal: client gets required information or the information material is sent to her/him
Category: primary
Precondition: -
Post condition success: client gets required information
Post condition failure: the required information can not be issued
Actors: client manager, client, company
Triggering event: client writes (letter, fax, e-mail) or calls
Description:
1. client data retrieval
2. information issue

Extension:
1. A client data actualization
2. A production of address label (for sending info-material)

Alternatives:
1. An inclusion of a new client

use case = sequence of actions

extend - relation

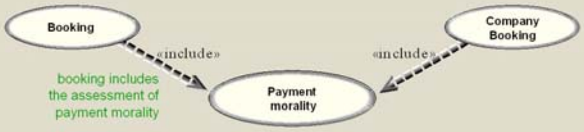


Basic use case

Extension use case

The customer requires mailing of information materials.

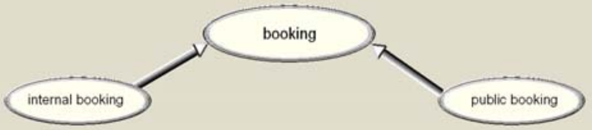
include - relation

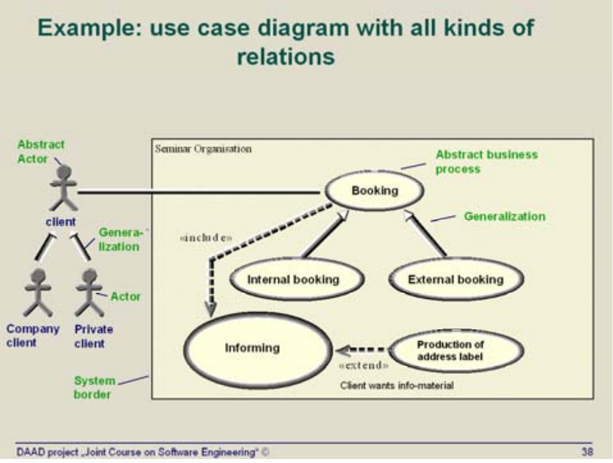


booking includes the assessment of payment morality

Generalize - relation

Booking is possible by companies and private persons.

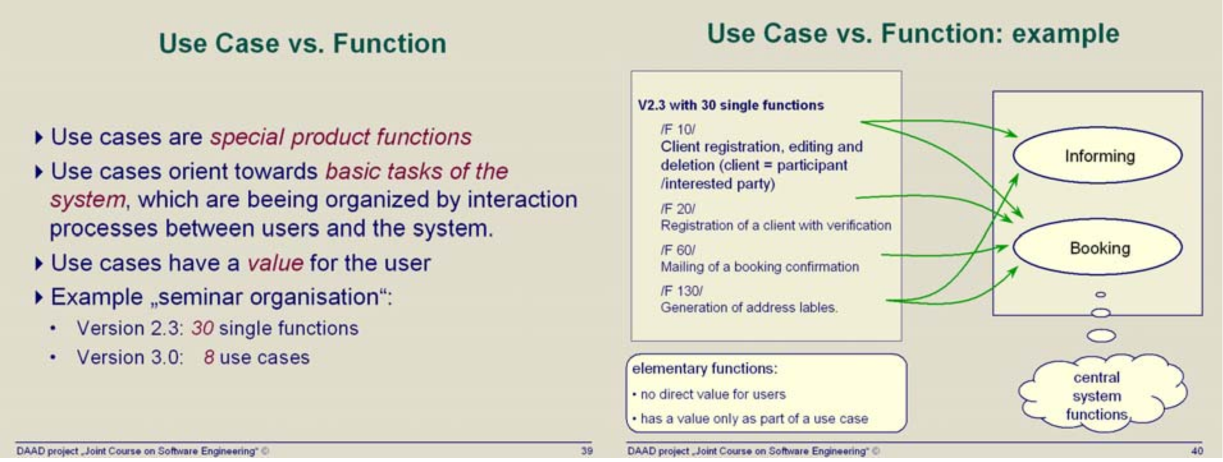




Example: use case diagram with all kinds of relations

DAAD project „Joint Course on Software Engineering" © 38

Slides comparing use cases with functions.



Use Case vs. Function

- Use cases are *special product functions*
- Use cases orient towards *basic tasks of the system*, which are beeing organized by interaction processes between users and the system.
- Use cases have a *value* for the user
- Example „seminar organisation":
  - Version 2.3: *30* single functions
  - Version 3.0: *8* use cases

DAAD project „Joint Course on Software Engineering" © 39

Use Case vs. Function: example

V2.3 with 30 single functions
/F 10/
Client registration, editing and deletion (client = participant /interested party)
/F 20/
Registration of a client with verification
/F 60/
Mailing of a booking confirmation
/F 130/
Generation of address lables.

elementary functions:
- no direct value for users
- has a value only as part of a use case

DAAD project „Joint Course on Software Engineering" © 40

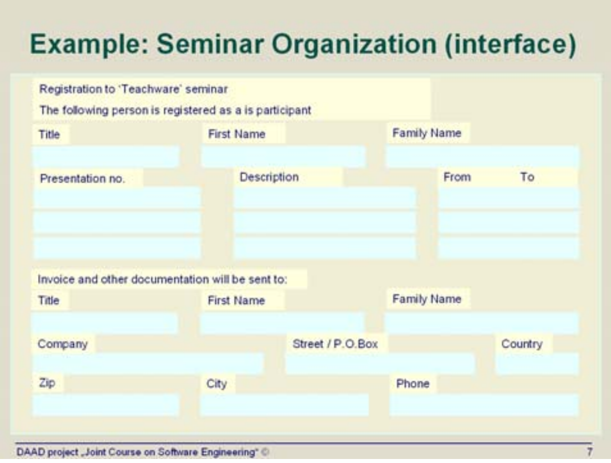
2.4 Topic 8: *Basic concepts of data oriented view*

Data dictionary and entity-relationship model should be illustrated with the case study.

To do:
- Design a user-interface form for which a relatively complex data dictionary can be created (see slides). Such data dictionary should illustrate most important data dictionary notations.
- Develop several entity-relationship diagrams from case study illustrating important entity-relationship notations (see slides).

Slides related to case study can be grouped in two groups:

Slides related to data dictionary.



Example: Seminar Organization (interface)

Registration to 'Teachware' seminar

The following person is registered as a is participant

Title

First Name

Family Name

Presentation no.

Description

From

To

Invoice and other documentation will be sent to:

Title

First Name

Family Name

Company

Street / P.O.Box

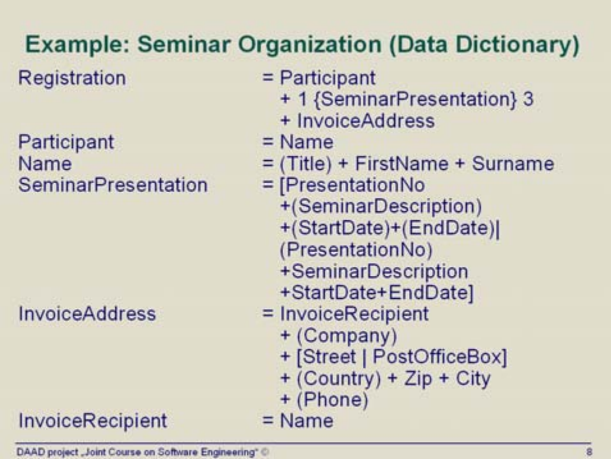
Country

Zip

City

Phone

DAAD project „Joint Course on Software Engineering" © 7

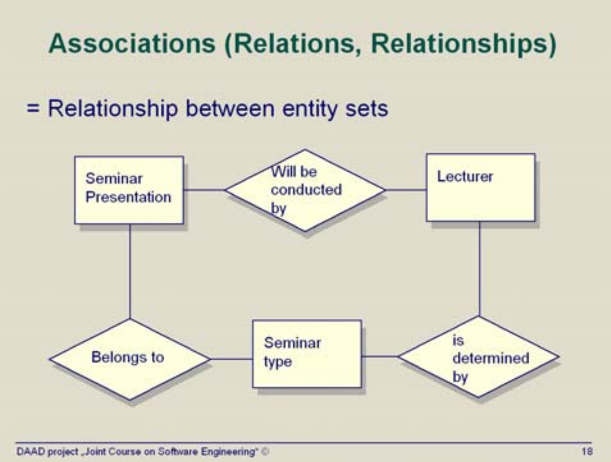


Example: Seminar Organization (Data Dictionary)

Registration	= Participant + 1 {SeminarPresentation} 3 + InvoiceAddress
Participant	= Name
Name	= (Title) + FirstName + Surname
SeminarPresentation	= [PresentationNo +(SeminarDescription) +(StartDate)+(EndDate)| (PresentationNo) +SeminarDescription +StartDate+EndDate]
InvoiceAddress	= InvoiceRecipient + (Company) + [Street | PostOfficeBox] + (Country) + Zip + City + (Phone)
InvoiceRecipient	= Name

DAAD project „Joint Course on Software Engineering" © 8

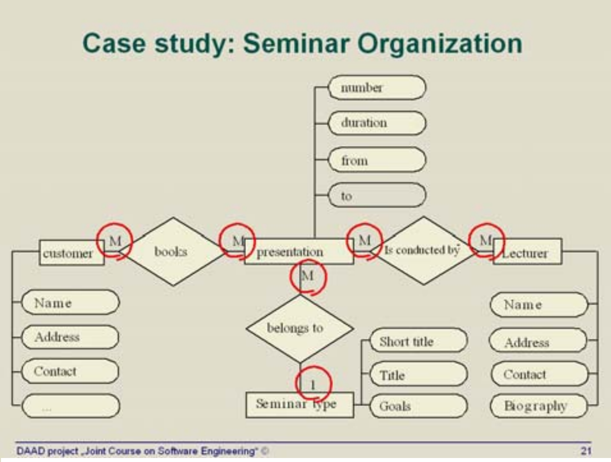
Slides related to entity-relationship model.



Associations (Relations, Relationships)

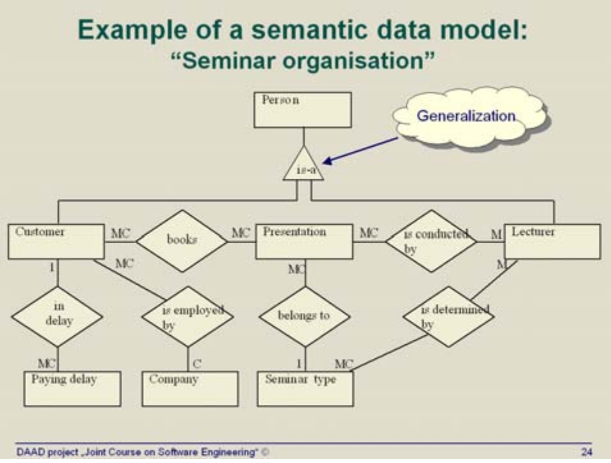
= Relationship between entity sets

DAAD project „Joint Course on Software Engineering" © 18



Case study: Seminar Organization

DAAD project „Joint Course on Software Engineering" © 21



Example of a semantic data model:
"Seminar organisation"

DAAD project „Joint Course on Software Engineering" © 24

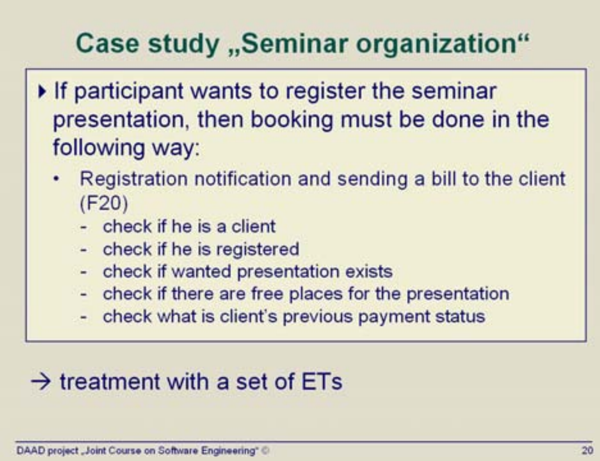
2.5 Topic 9: *Basic concepts of the rule oriented view*

Decision tables should be illustrated with the case study.

To do:

- Develop an example suitable for description with decision tables (checks, rules, sequence of activities that can be done under certain conditions, …) (see slides).

All slides related to case study belong only to one group:
Slides illustrating decision tables.

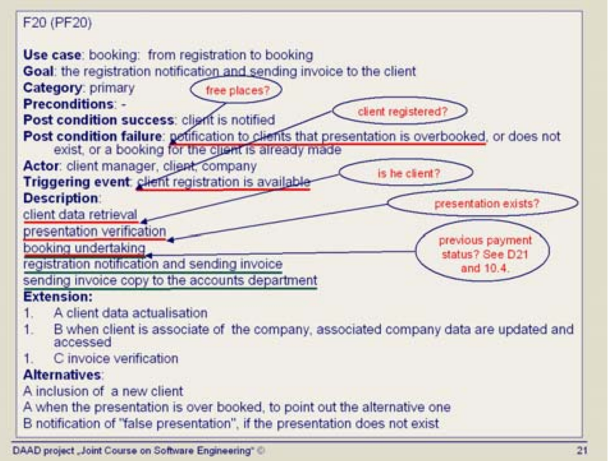


Case study „Seminar organization“

- If participant wants to register the seminar presentation, then booking must be done in the following way:
  - Registration notification and sending a bill to the client (F20)
    - check if he is a client
    - check if he is registered
    - check if wanted presentation exists
    - check if there are free places for the presentation
    - check what is client's previous payment status

→ treatment with a set of ETs

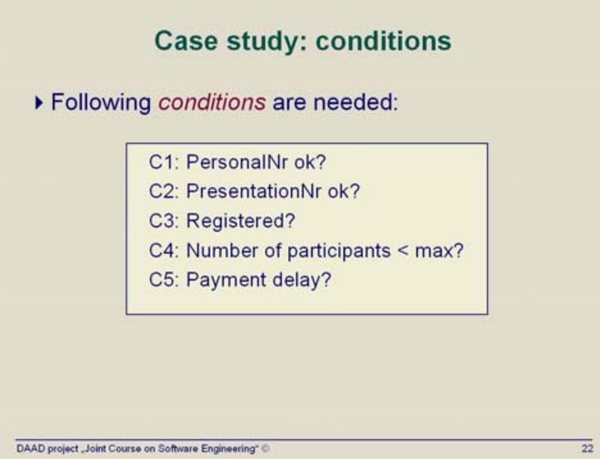
DAAD project „Joint Course on Software Engineering“ © 20



F20 (PF20)

Use case: booking: from registration to booking
Goal: the registration notification and sending invoice to the client
Category: primary
Preconditions: -
Post condition success: client is notified
Post condition failure: notification to clients that presentation is overbooked, or does not exist, or a booking for the client is already made
Actor: client manager, client, company
Triggering event: client registration is available
Description:
client data retrieval
presentation verification
booking undertaking
registration notification and sending invoice
sending invoice copy to the accounts department
Extension:
1. A client data actualisation
1. B when client is associate of the company, associated company data are updated and accessed
1. C invoice verification
Alternatives:
A inclusion of a new client
A when the presentation is over booked, to point out the alternative one
B notification of "false presentation", if the presentation does not exist

DAAD project „Joint Course on Software Engineering“ © 21

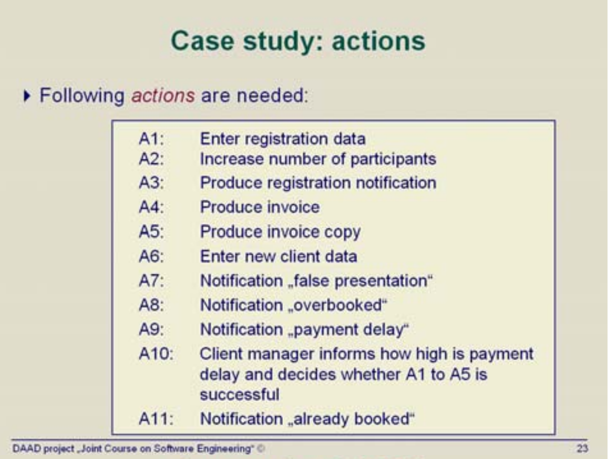


Case study: conditions

- Following *conditions* are needed:

C1: PersonalNr ok?
C2: PresentationNr ok?
C3: Registered?
C4: Number of participants < max?
C5: Payment delay?

DAAD project „Joint Course on Software Engineering“ © 22

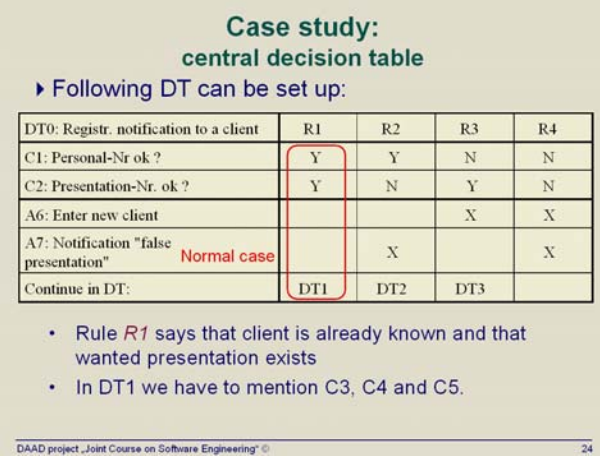


Case study: actions

- Following *actions* are needed:

A1:	Enter registration data
A2:	Increase number of participants
A3:	Produce registration notification
A4:	Produce invoice
A5:	Produce invoice copy
A6:	Enter new client data
A7:	Notification „false presentation“
A8:	Notification „overbooked“
A9:	Notification „payment delay“
A10:	Client manager informs how high is payment delay and decides whether A1 to A5 is successful
A11:	Notification „already booked“

DAAD project „Joint Course on Software Engineering“ © 23



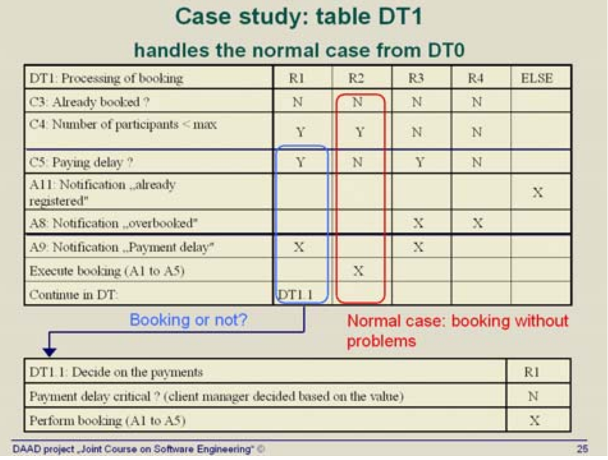
Case study:
central decision table

- Following DT can be set up:

DT0: Registr. notification to a client	R1	R2	R3	R4
C1: Personal-Nr ok ?	Y	Y	N	N
C2: Presentation-Nr. ok ?	Y	N	Y	N
A6: Enter new client			X	X
A7: Notification "false presentation" Normal case		X		X
Continue in DT:	DT1	DT2	DT3	

- Rule *R1* says that client is already known and that wanted presentation exists
- In DT1 we have to mention C3, C4 and C5.

DAAD project „Joint Course on Software Engineering“ © 24



Case study: table DT1
handles the normal case from DT0

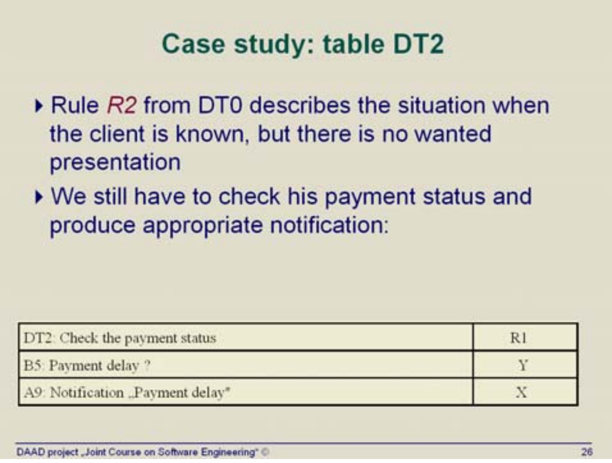
DT1: Processing of booking	R1	R2	R3	R4	ELSE
C3: Already booked ?	N	N	N	N	
C4: Number of participants < max	Y	Y	N	N	
C5: Paying delay ?	Y	N	Y	N	
A11: Notification „already registered"					X
A8: Notification „overbooked"			X	X	
A9: Notification „Payment delay"	X		X		
Execute booking (A1 to A5)		X			
Continue in DT:	DT1.1				

Booking or not?

Normal case: booking without problems

DT1.1: Decide on the payments	R1
Payment delay critical ? (client manager decided based on the value)	N
Perform booking (A1 to A5)	X

DAAD project „Joint Course on Software Engineering“ © 25

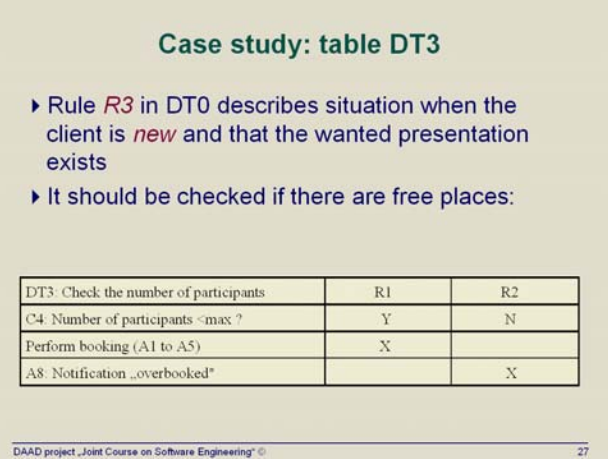


Case study: table DT2

- Rule *R2* from DT0 describes the situation when the client is known, but there is no wanted presentation
- We still have to check his payment status and produce appropriate notification:

DT2: Check the payment status	R1
B5: Payment delay ?	Y
A9: Notification „Payment delay"	X

DAAD project „Joint Course on Software Engineering" © 26



Case study: table DT3

- Rule *R3* in DT0 describes situation when the client is *new* and that the wanted presentation exists
- It should be checked if there are free places:

DT3: Check the number of participants	R1	R2
C4: Number of participants <max ?	Y	N
Perform booking (A1 to A5)	X	
A8: Notification „overbooked"		X

DAAD project „Joint Course on Software Engineering" © 27

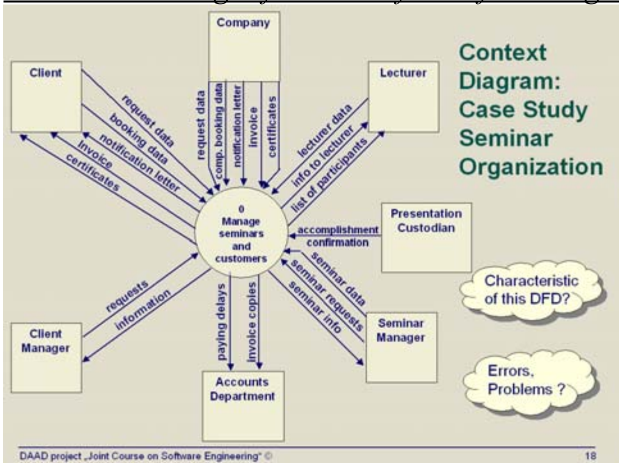
2.6 Topic 10: *Structured analysis*

This topic is essentially driven by the data-flow diagrams developed for the case study. They are gradually refined, and finished with mini-specifications and data dictionaries. This topic also presents once more an already developed function tree that is implicitly contained in diagrams.

To do:
- Develop an example for mini-specification for at least one illustrative data flow diagram.
- Develop an example for a data dictionary that will be used with refinement of data flow diagrams.

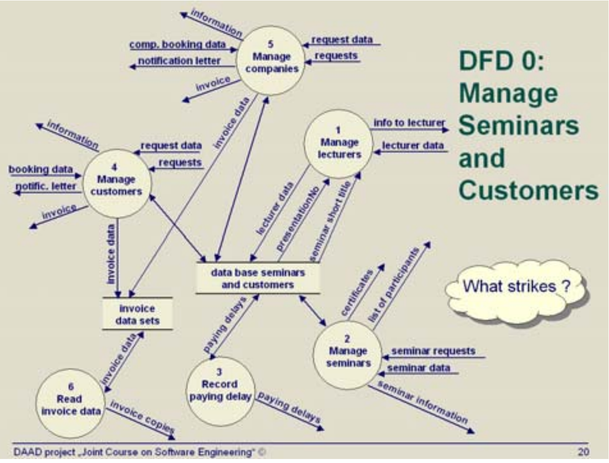
All slides related to case study belong to three groups:

Slides illustrating refinement of data flow diagrams.



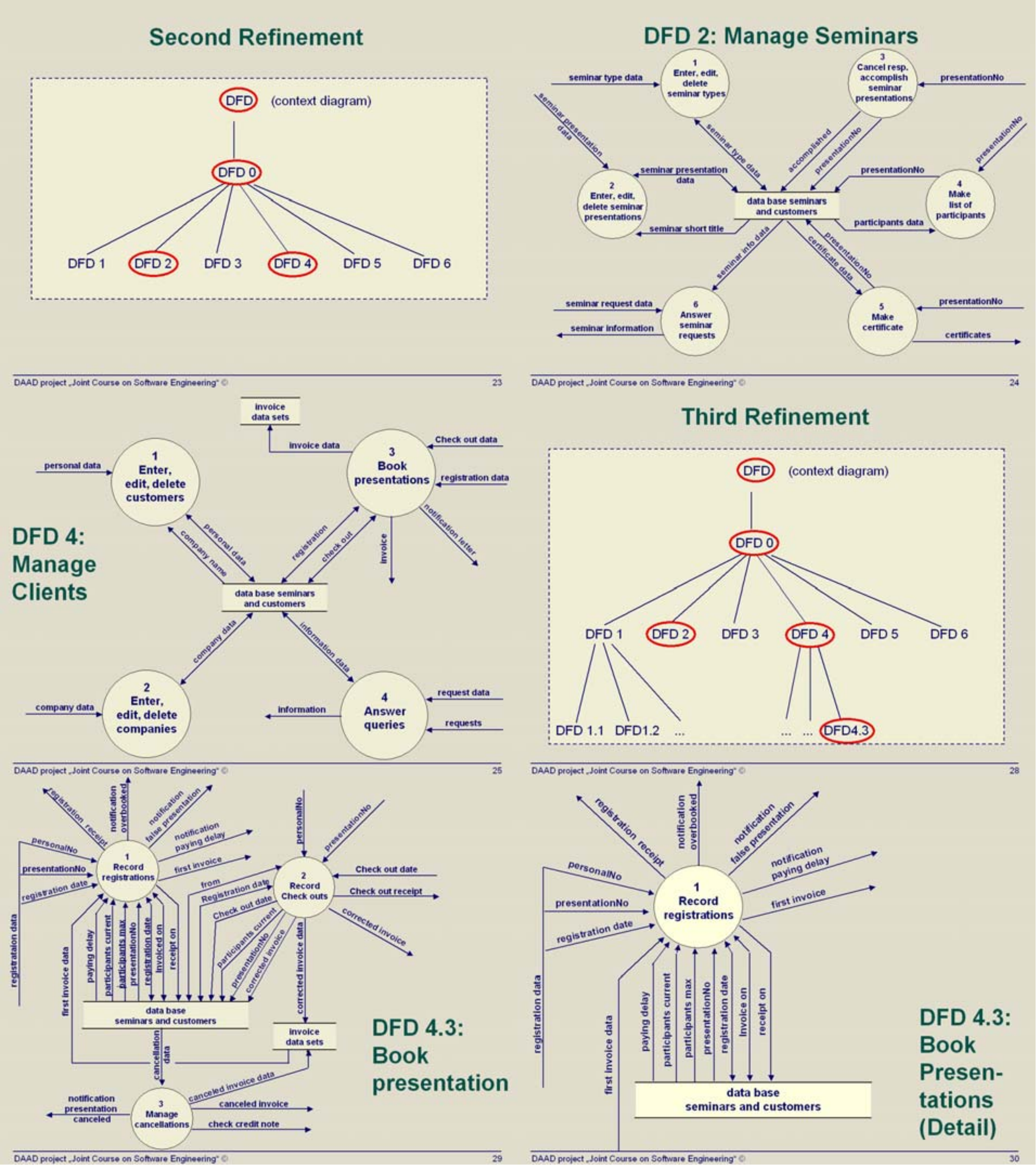
Context Diagram: Case Study Seminar Organization

DAAD project „Joint Course on Software Engineering" © 18



DFD 0: Manage Seminars and Customers

DAAD project „Joint Course on Software Engineering" © 20



Second Refinement

DAAD project „Joint Course on Software Engineering" © 23

DFD 2: Manage Seminars

DAAD project „Joint Course on Software Engineering" © 24

DFD 4: Manage Clients

DAAD project „Joint Course on Software Engineering" © 25

Third Refinement

DAAD project „Joint Course on Software Engineering" © 28

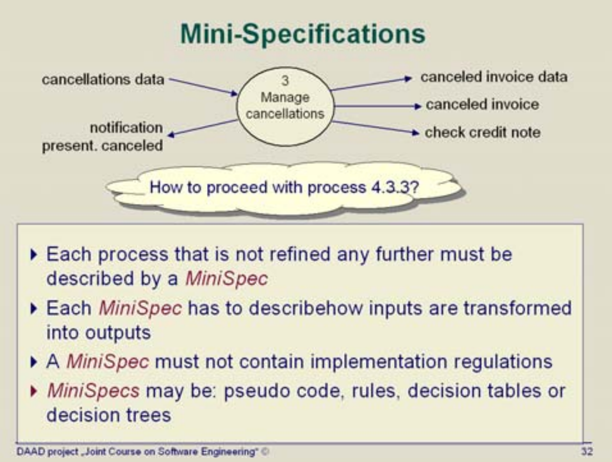
DFD 4.3: Book presentation

DAAD project „Joint Course on Software Engineering" © 29

DFD 4.3: Book Presen-tations (Detail)

DAAD project „Joint Course on Software Engineering" © 30

Slides illustrating mini-specifications (and function tree).

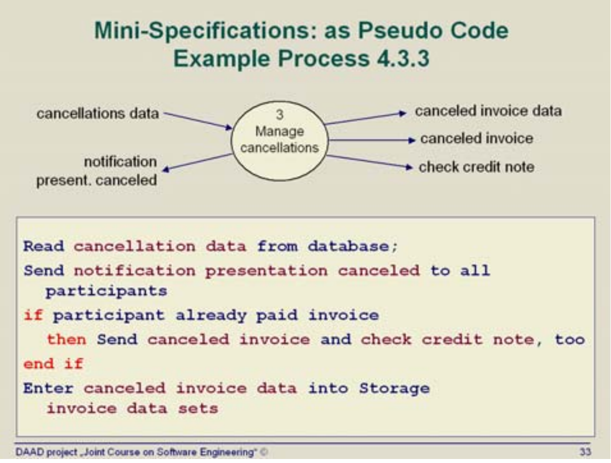


Mini-Specifications

cancellations data
3
Manage
cancellations
canceled invoice data
canceled invoice
check credit note
notification
present. canceled
How to proceed with process 4.3.3?

- Each process that is not refined any further must be described by a *MiniSpec*
- Each *MiniSpec* has to describehow inputs are transformed into outputs
- A *MiniSpec* must not contain implementation regulations
- *MiniSpecs* may be: pseudo code, rules, decision tables or decision trees

DAAD project „Joint Course on Software Engineering" © 32

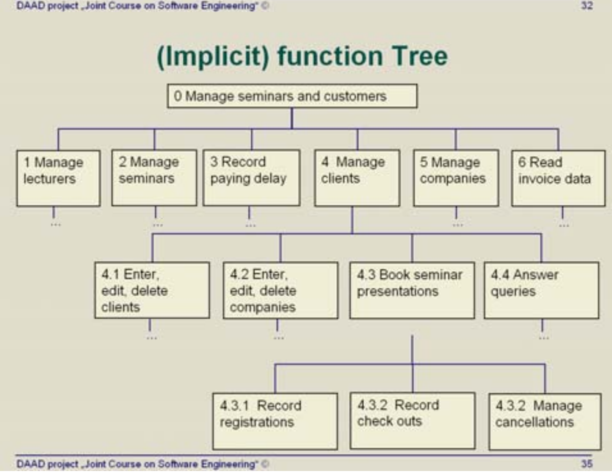


Mini-Specifications: as Pseudo Code Example Process 4.3.3

cancellations data
3
Manage
cancellations
canceled invoice data
canceled invoice
check credit note
notification
present. canceled

```
Read cancellation data from database;
Send notification presentation canceled to all
  participants
if participant already paid invoice
  then Send canceled invoice and check credit note, too
end if
Enter canceled invoice data into Storage
  invoice data sets
```

DAAD project „Joint Course on Software Engineering" © 33

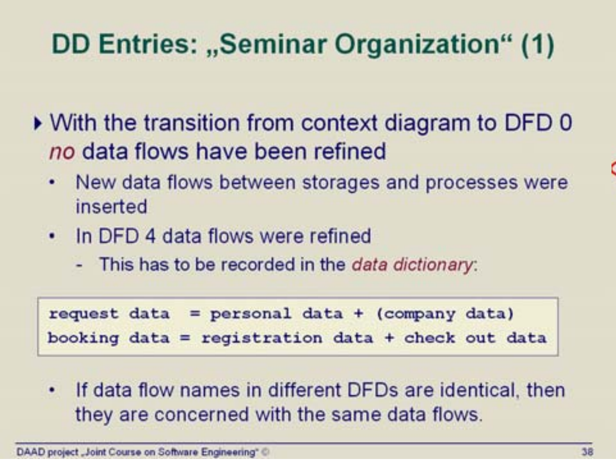


(Implicit) function Tree

0 Manage seminars and customers
1 Manage lecturers
2 Manage seminars
3 Record paying delay
4 Manage clients
5 Manage companies
6 Read invoice data
4.1 Enter, edit, delete clients
4.2 Enter, edit, delete companies
4.3 Book seminar presentations
4.4 Answer queries
4.3.1 Record registrations
4.3.2 Record check outs
4.3.2 Manage cancellations
...

DAAD project „Joint Course on Software Engineering" © 35

<u>Slides illustrating data dictionary and its use with data-flow diagrams</u>.



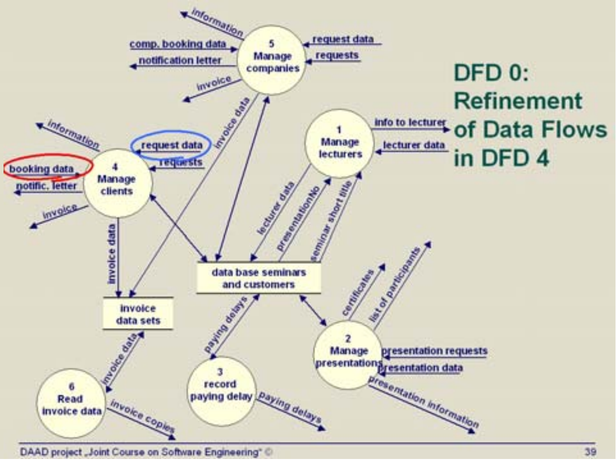
DD Entries: „Seminar Organization" (1)

- With the transition from context diagram to DFD 0 *no* data flows have been refined
  - New data flows between storages and processes were inserted
  - In DFD 4 data flows were refined
    - This has to be recorded in the *data dictionary*:

```
request data  = personal data + (company data)
booking data = registration data + check out data
```

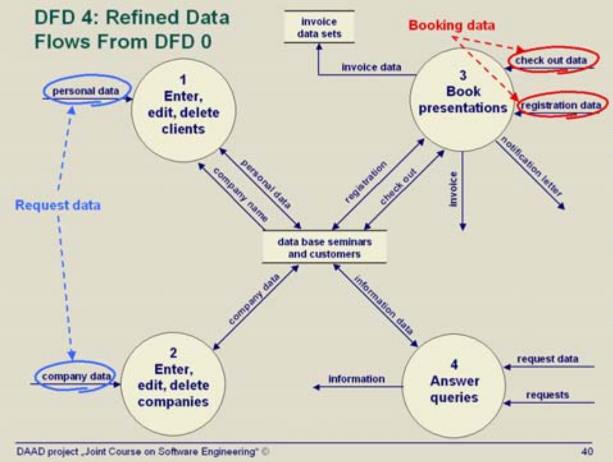
  - If data flow names in different DFDs are identical, then they are concerned with the same data flows.

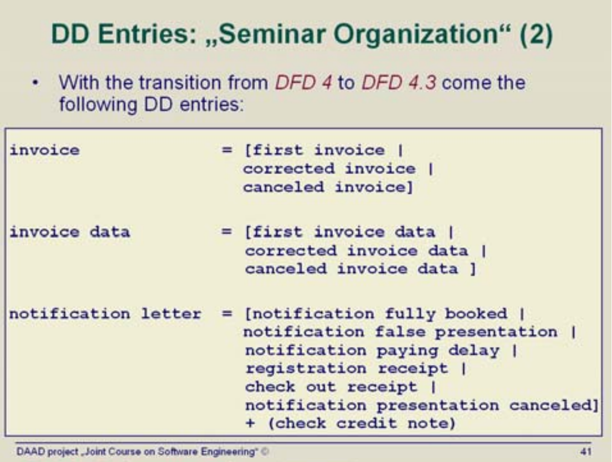
DAAD project „Joint Course on Software Engineering" © 38



DFD 0: Refinement of Data Flows in DFD 4

DAAD project „Joint Course on Software Engineering" © 39

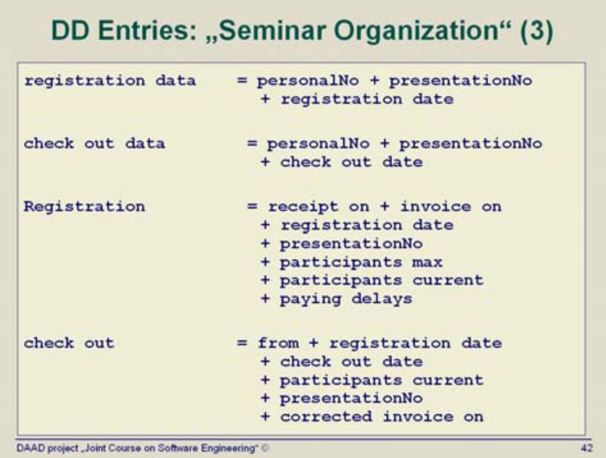

DFD 4: Refined Data Flows From DFD 0


DD Entries: „Seminar Organization“ (2)

- With the transition from *DFD 4* to *DFD 4.3* come the following DD entries:

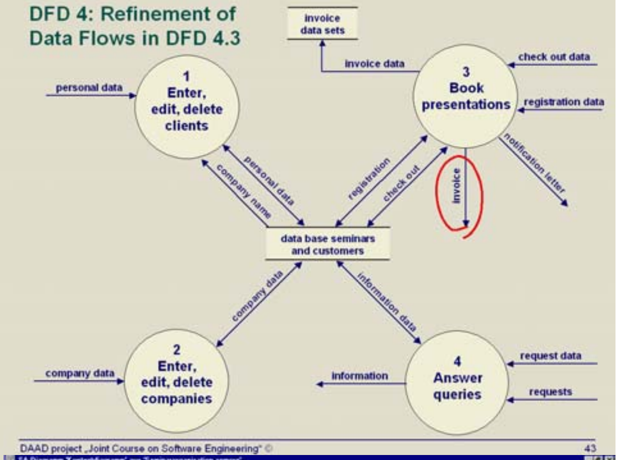
```
invoice              = [first invoice |
                        corrected invoice |
                        canceled invoice]

invoice data         = [first invoice data |
                        corrected invoice data |
                        canceled invoice data ]

notification letter  = [notification fully booked |
                        notification false presentation |
                        notification paying delay |
                        registration receipt |
                        check out receipt |
                        notification presentation canceled]
                        + (check credit note)
```

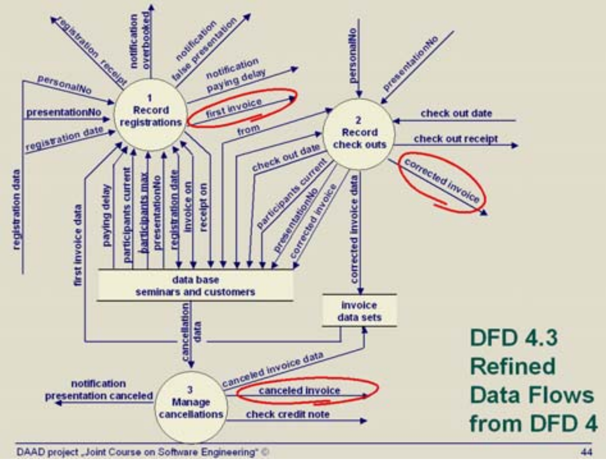
DAAD project „Joint Course on Software Engineering“ © 41

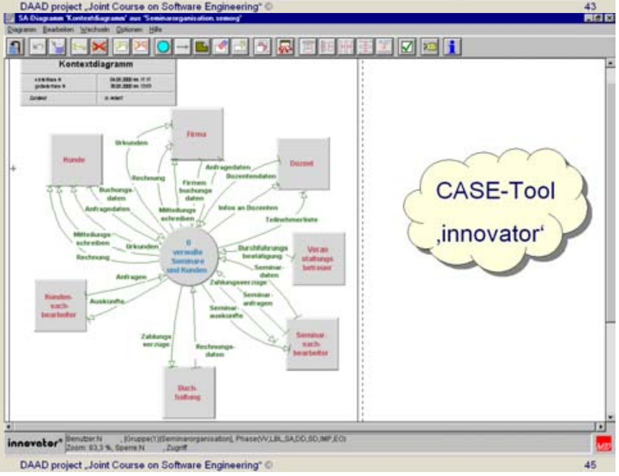

DD Entries: „Seminar Organization“ (3)

```
registration data    = personalNo + presentationNo
                       + registration date

check out data        = personalNo + presentationNo
                       + check out date

Registration          = receipt on + invoice on
                       + registration date
                       + presentationNo
                       + participants max
                       + participants current
                       + paying delays

check out            = from + registration date
                       + check out date
                       + participants current
                       + presentationNo
                       + corrected invoice on
```

DAAD project „Joint Course on Software Engineering“ © 42


DFD 4: Refinement of Data Flows in DFD 4.3


DFD 4.3 Refined Data Flows from DFD 4



2.7 Topic 11: *Basic concepts of state oriented view*

The topic uses case study to illustrate notational elements of automata and activity diagram. Also, two slides of with excerpts from two CASE tools are given.

To do:
- Develop an example for an object life cycle that can illustrate important notational aspects (see slides).

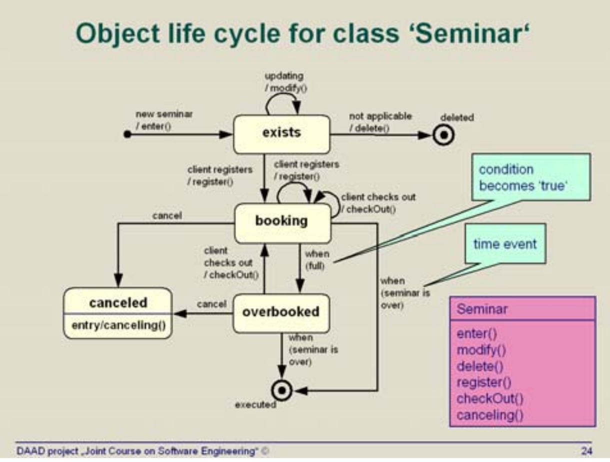
- Develop an example for an activity diagram that can illustrate important notational aspects (see slide).

Remark.

Additional slides can be also produced.

All slides related to case study belong to three groups:

Slides illustrating object life cycle.

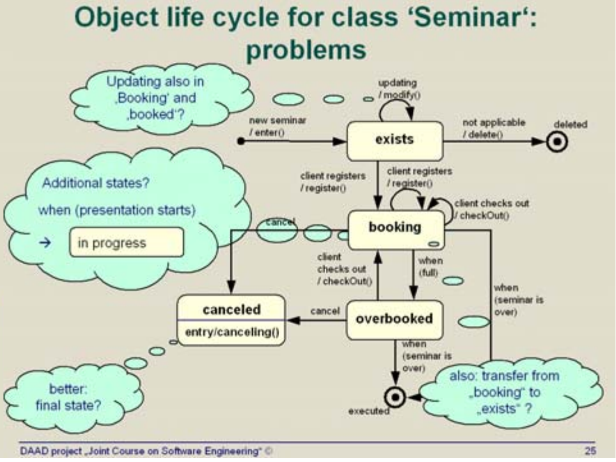


Object life cycle for class 'Seminar'

condition becomes 'true'

time event

DAAD project „Joint Course on Software Engineering" © 24



Object life cycle for class 'Seminar': problems

Updating also in 'Booking' and 'booked'?

Additional states?

when (presentation starts)

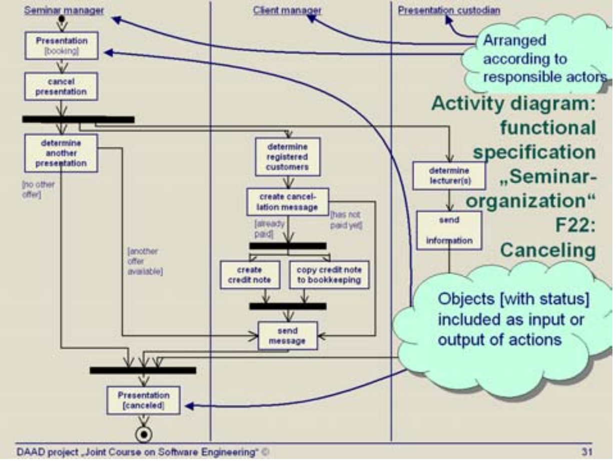
→ in progress

better: final state?

also: transfer from „booking" to „exists" ?

DAAD project „Joint Course on Software Engineering" © 25

Slide illustrating activity diagram.



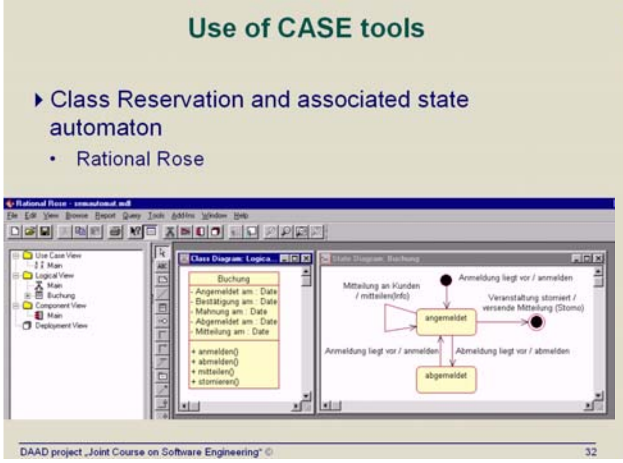
Activity diagram: functional specification „Seminar-organization" F22: Canceling

Arranged according to responsible actors

Objects [with status] included as input or output of actions

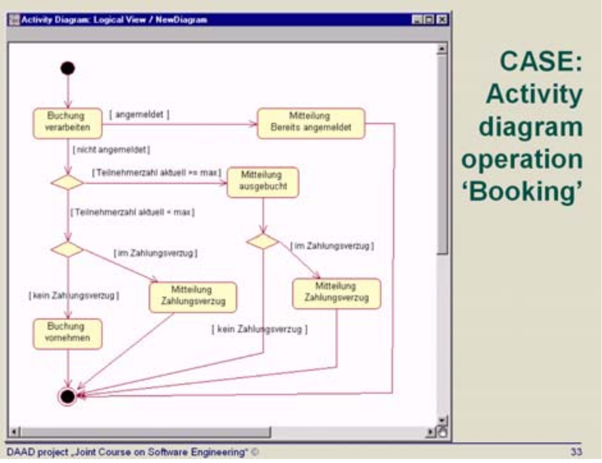
DAAD project „Joint Course on Software Engineering" © 31

Slides illustrating CASE tools.


Use of CASE tools

- Class Reservation and associated state automaton
  - Rational Rose

DAAD project „Joint Course on Software Engineering" © 32


CASE: Activity diagram operation 'Booking'

Activity Diagram: Logical View / NewDiagram
Buchung verarbeiten
[angemeldet]
Mitteilung Bereits angemeldet
[nicht angemeldet]
[Teilnehmerzahl aktuell >= max]
Mitteilung ausgebucht
[Teilnehmerzahl aktuell < max]
[im Zahlungsverzug]
[im Zahlungsverzug]
[kein Zahlungsverzug]
Mitteilung Zahlungsverzug
Mitteilung Zahlungsverzug
Buchung vornehmen
[kein Zahlungsverzug]

DAAD project „Joint Course on Software Engineering" © 33

2.8 Topic 12: *Basic concepts of scenario-based view*

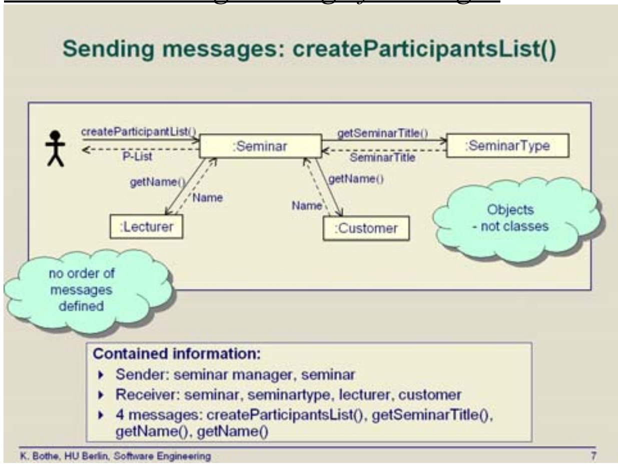
The topic uses case study to illustrate principles of sending messages and notational elements of collaboration diagram and sequence diagram.

To do:
- Develop an example that can be used to illustrate the principle of sending messages in OO world. (see slides).
- Develop an example for a collaboration diagram that can illustrate important notational aspects (see slide).
- Develop an example for a sequence diagram that can illustrate important notational aspects (see slide).

Remark.
Additional slides can be also produced.

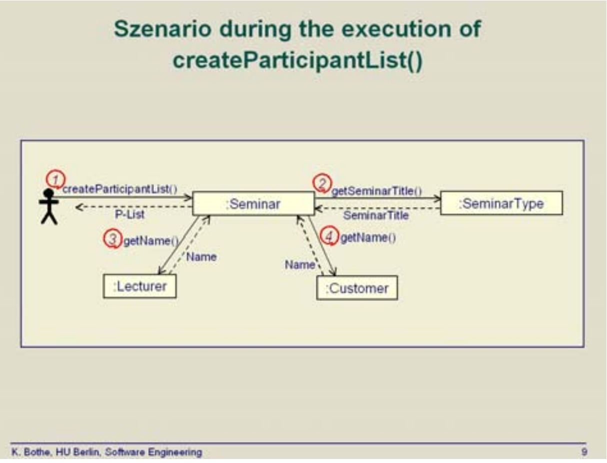
All slides related to case study belong to two groups:

Slides illustrating sending of messages.


Sending messages: createParticipantsList()

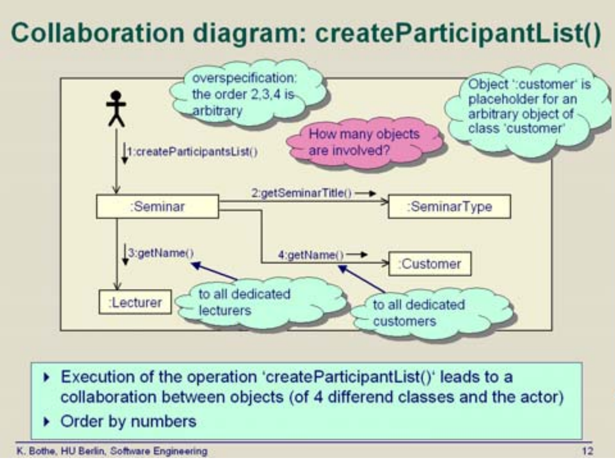
Contained information:
- Sender: seminar manager, seminar
- Receiver: seminar, seminartype, lecturer, customer
- 4 messages: createParticipantsList(), getSeminarTitle(), getName(), getName()

K. Bothe, HU Berlin, Software Engineering 7


Szenario during the execution of createParticipantList()

K. Bothe, HU Berlin, Software Engineering 9

Slides illustrating diagrams.

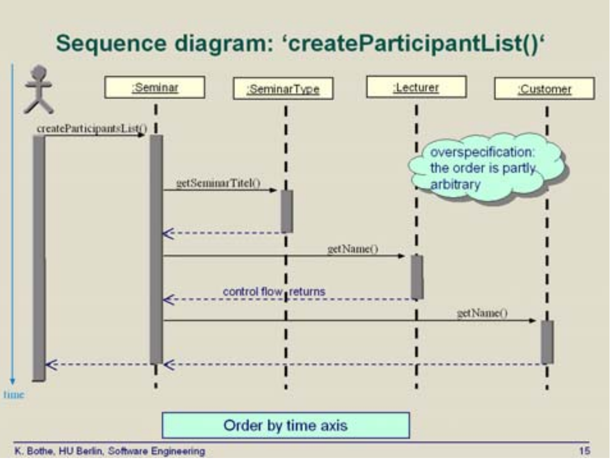


Collaboration diagram: createParticipantList()

overspecification: the order 2,3,4 is arbitrary
Object ':customer' is placeholder for an arbitrary object of class 'customer'
How many objects are involved?
1:createParticipantsList()
:Seminar
2:getSeminarTitle()
:SeminarType
3:getName()
4:getName()
:Customer
:Lecturer
to all dedicated lecturers
to all dedicated customers

- Execution of the operation 'createParticipantList()' leads to a collaboration between objects (of 4 differend classes and the actor)
- Order by numbers

K. Bothe, HU Berlin, Software Engineering 12



Sequence diagram: 'createParticipantList()'

Order by time axis

K. Bothe, HU Berlin, Software Engineering 15

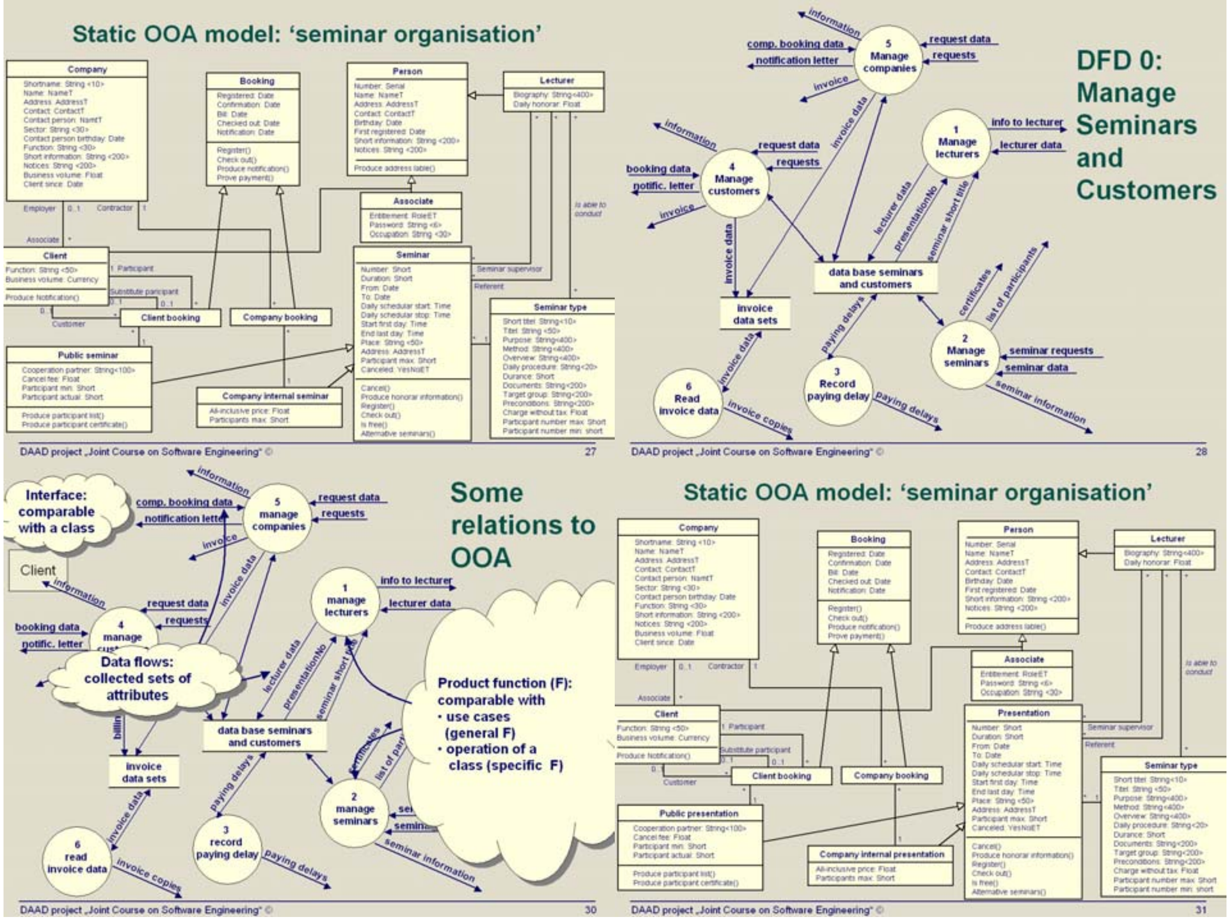
2.9 Topic 13: *Object-oriented analysis*

This topic uses case study to introduce all notational aspects of class diagrams and to elaborate on possible other in designing classes. Topic also uses already presented slides form *Topic 10: Structured Analysis.*

To do:
- Develop a *full* class diagram for a case study, in such a way that elaboration of other possibilities (choice of classes, etc.) can be done.

All slides related to case study belong to three groups:

Slides comparing OOA with SA.



Static OOA model: ‘seminar organisation’

Company
Shortname: String <10>
Name: NameT
Address: AddressT
Contact: ContactT
Contact person: NamtT
Sector: String <30>
Contact person birthday: Date
Function: String <30>
Short information: String <200>
Notices: String <200>
Business volume: Float
Client since: Date

Booking
Registered: Date
Confirmation: Date
Bill: Date
Checked out: Date
Notification: Date
Register()
Check out()
Produce notification()
Prove payment()

Person
Number: Serial
Name: NameT
Address: AddressT
Contact: ContactT
Birthday: Date
First registered: Date
Short information: String <200>
Notices: String <200>
Produce address lable()

Lecturer
Biography: String<400>
Daily honorar: Float

Associate
Entitlement: RoleET
Password: String <6>
Occupation: String <30>

Client
Function: String <50>
Business volume: Currency
Produce Notification()

Seminar
Number: Short
Duration: Short
From: Date
To: Date
Daily schedular start: Time
Daily schedular stop: Time
Start first day: Time
End last day: Time
Place: String <50>
Address: AddressT
Participant max: Short
Canceled: YesNoET
Cancel()
Produce honorar information()
Register()
Check out()
Is free()
Alternative seminars()

Seminar type
Short titel: String<10>
Titel: String <50>
Purpose: String<400>
Method: String<400>
Overview: String<400>
Daily procedure: String<20>
Durance: Short
Documents: String<200>
Target group: String<200>
Preconditions: String<200>
Charge without tax: Float
Participant number max: Short
Participant number min: short

Public seminar
Cooperation partner: String<100>
Cancel fee: Float
Participant min: Short
Participant actual: Short
Produce participant list()
Produce participant certificate()

Company internal seminar
All-inclusive price: Float
Participants max: Short

Client booking

Company booking

DAAD project „Joint Course on Software Engineering“ © 27

DFD 0: Manage Seminars and Customers

DAAD project „Joint Course on Software Engineering“ © 28

Some relations to OOA

Interface: comparable with a class

Client

Data flows: collected sets of attributes

Product function (F): comparable with
- use cases (general F)
- operation of a class (specific F)

DAAD project „Joint Course on Software Engineering“ © 30

Static OOA model: ‘seminar organisation’

Company
Shortname: String <10>
Name: NameT
Address: AddressT
Contact: ContactT
Contact person: NamtT
Sector: String <30>
Contact person birthday: Date
Function: String <30>
Short information: String <200>
Notices: String <200>
Business volume: Float
Client since: Date

Booking
Registered: Date
Confirmation: Date
Bill: Date
Checked out: Date
Notification: Date
Register()
Check out()
Produce notification()
Prove payment()

Person
Number: Serial
Name: NameT
Address: AddressT
Contact: ContactT
Birthday: Date
First registered: Date
Short information: String <200>
Notices: String <200>
Produce address lable()

Lecturer
Biography: String<400>
Daily honorar: Float

Associate
Entitlement: RoleET
Password: String <6>
Occupation: String <30>

Client
Function: String <50>
Business volume: Currency
Produce Notification()

Presentation
Number: Short
Duration: Short
From: Date
To: Date
Daily schedular start: Time
Daily schedular stop: Time
Start first day: Time
End last day: Time
Place: String <50>
Address: AddressT
Participant max: Short
Canceled: YesNoET
Cancel()
Produce honorar information()
Register()
Check out()
Is free()
Alternative seminars()

Seminar type
Short titel: String<10>
Titel: String <50>
Purpose: String<400>
Method: String<400>
Overview: String<400>
Daily procedure: String<20>
Durance: Short
Documents: String<200>
Target group: String<200>
Preconditions: String<200>
Charge without tax: Float
Participant number max: Short
Participant number min: short

Public presentation
Cooperation partner: String<100>
Cancel fee: Float
Participant min: Short
Participant actual: Short
Produce participant list()
Produce participant certificate()

Company internal presentation
All-inclusive price: Float
Participants max: Short

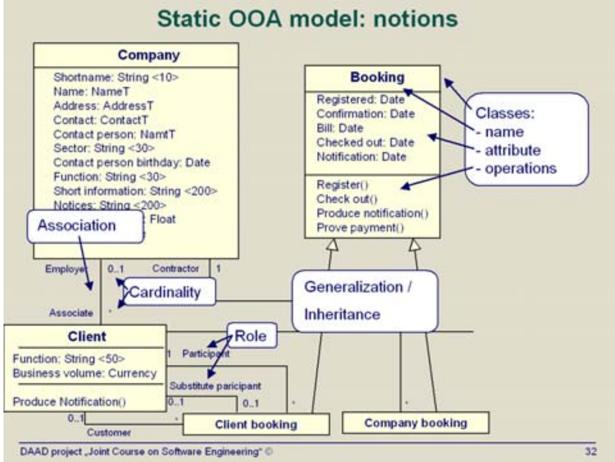
Client booking

Company booking

DAAD project „Joint Course on Software Engineering“ © 31

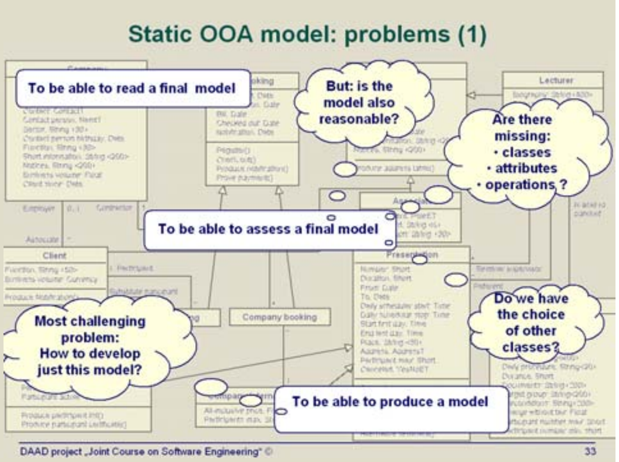
Slides introducing notations and problems (other possibilities) of class diagrams.

Static OOA model: notions



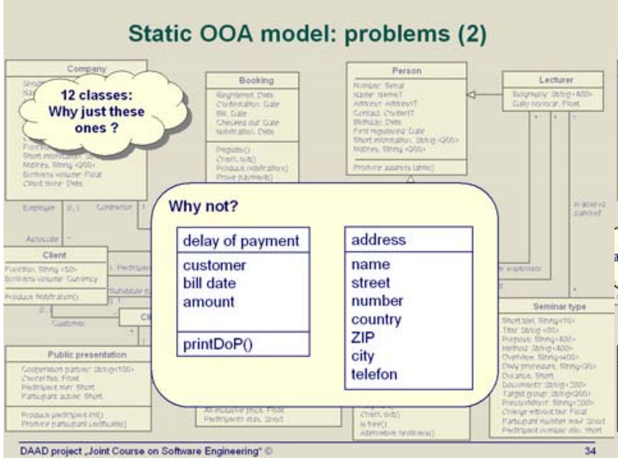
DAAD project „Joint Course on Software Engineering“ © 32

Static OOA model: problems (1)


To be able to read a final model
But: is the model also reasonable?
Are there missing:
• classes
• attributes
• operations ?
To be able to assess a final model
Lecturer
Client
Presentation
Company booking
Most challenging problem:
How to develop just this model?
Do we have the choice of other classes?
To be able to produce a model

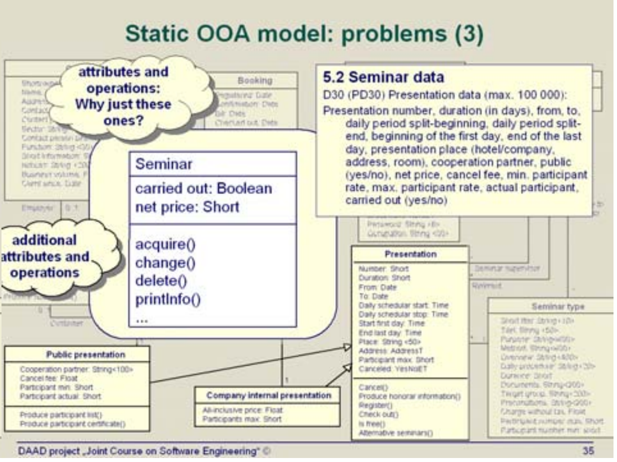
DAAD project „Joint Course on Software Engineering“ © 33

Static OOA model: problems (2)



DAAD project „Joint Course on Software Engineering“ © 34

Static OOA model: problems (3)

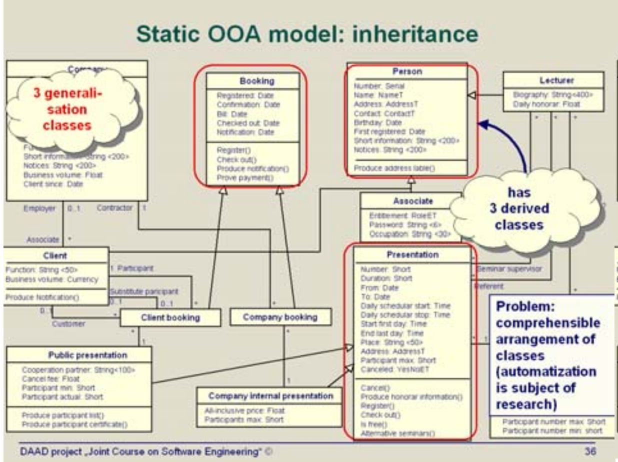


5.2 Seminar data

D30 (PD30) Presentation data (max. 100 000):

Presentation number, duration (in days), from, to, daily period split-beginning, daily period split-end, beginning of the first day, end of the last day, presentation place (hotel/company, address, room), cooperation partner, public (yes/no), net price, cancel fee, min. participant rate, max. participant rate, actual participant, carried out (yes/no)

DAAD project „Joint Course on Software Engineering“ © 35



Static OOA model: inheritance

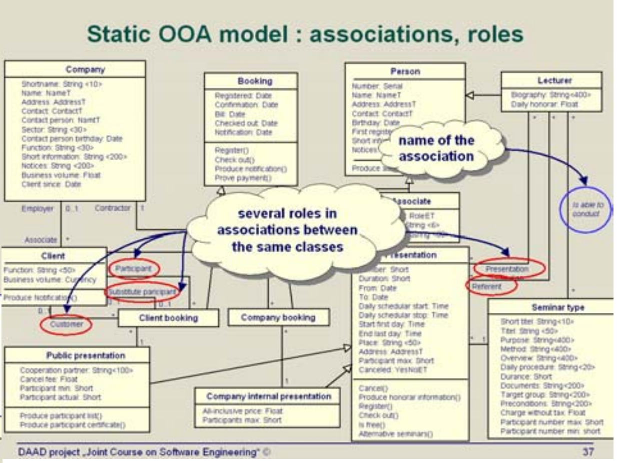
3 generalisation classes

has 3 derived classes

Problem: comprehensible arrangement of classes (automatization is subject of research)

DAAD project „Joint Course on Software Engineering“ ©

36



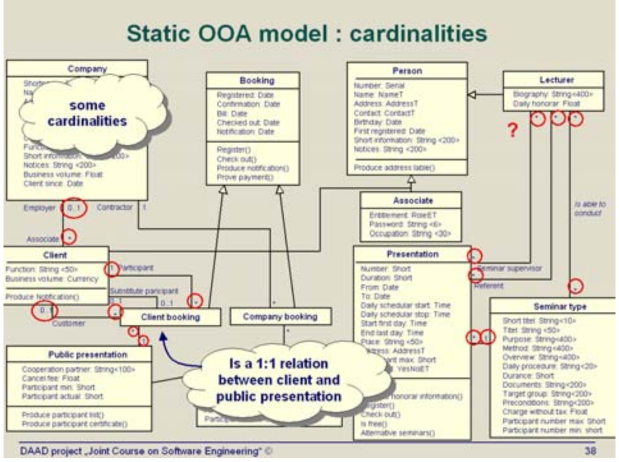
Static OOA model : associations, roles

name of the association

several roles in associations between the same classes

DAAD project „Joint Course on Software Engineering“ ©

37



Static OOA model : cardinalities

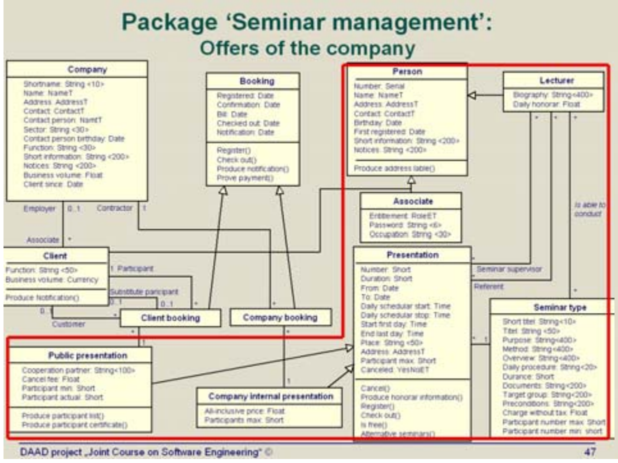
some cardinalities

Is a 1:1 relation between client and public presentation

DAAD project „Joint Course on Software Engineering“ ©

38

Slides illustrating packages.

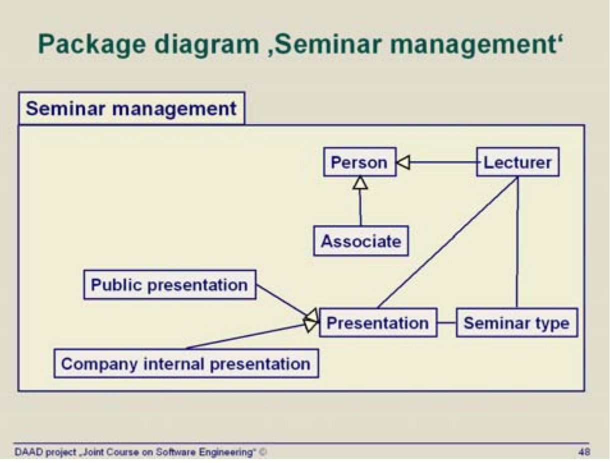


Package 'Seminar management':

Offers of the company

DAAD project „Joint Course on Software Engineering“ ©

47



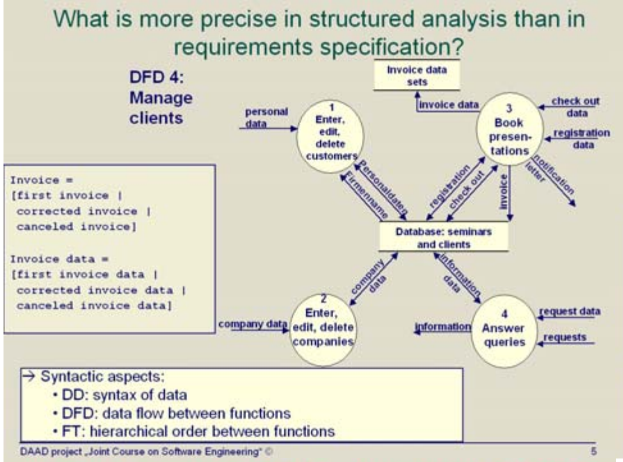
Package diagram ‚Seminar management‘

DAAD project „Joint Course on Software Engineering“ ©

48

2.10 Topic 14: *Formal specifications*

Parts of already developed documents/slides are used here in order to clarify the need for formal specifications. No special effort is neede here, except to use and slighly adapt already existing slides.

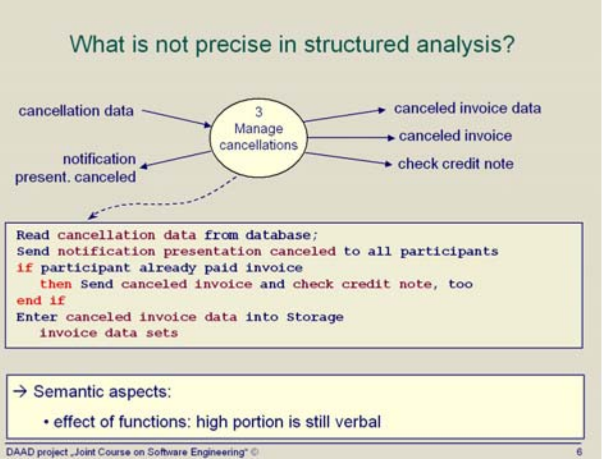


What is more precise in structured analysis than in requirements specification?

```
Invoice =
[first invoice |
 corrected invoice |
 canceled invoice]

Invoice data =
[first invoice data |
 corrected invoice data |
 canceled invoice data]
```

→ Syntactic aspects:
- DD: syntax of data
- DFD: data flow between functions
- FT: hierarchical order between functions

DAAD project „Joint Course on Software Engineering" © 5

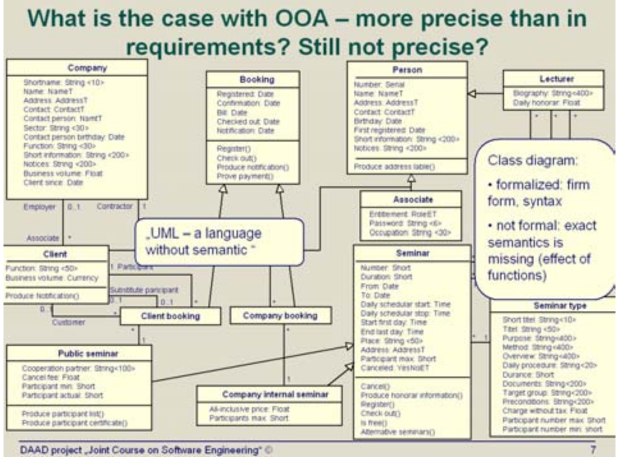


What is not precise in structured analysis?

```
Read cancellation data from database;
Send notification presentation canceled to all participants
if participant already paid invoice
   then Send canceled invoice and check credit note, too
end if
Enter canceled invoice data into Storage
   invoice data sets
```

→ Semantic aspects:
- effect of functions: high portion is still verbal

DAAD project „Joint Course on Software Engineering" © 6



What is the case with OOA – more precise than in requirements? Still not precise?

„UML – a language without semantic "

Class diagram:
- formalized: firm form, syntax
- not formal: exact semantics is missing (effect of functions)

DAAD project „Joint Course on Software Engineering" © 7

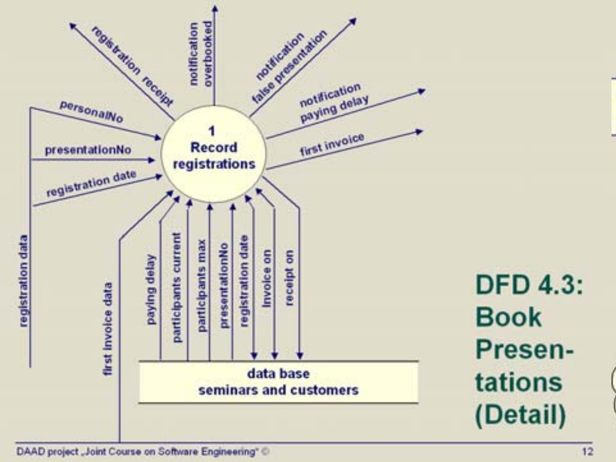
2.11 Topic 16: *Structured design*

This topic uses case study to introduce basic principles of structured design. It takes one data-flow diagram and proceeds in designing the software for it.

To do:
- Develop a structured design for at least one data-flow diagram from structured analysis..

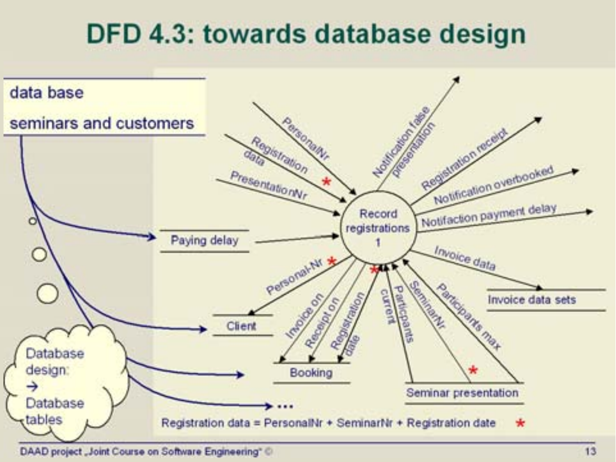
All slides related to case study belong to one group:

Slides introducing notations and principles of structured design.



DFD 4.3: Book Presentations (Detail)

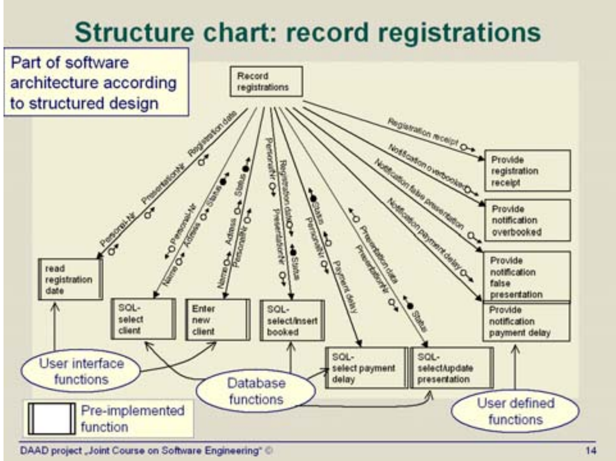
DAAD project „Joint Course on Software Engineering“ © 12



DFD 4.3: towards database design

Registration data = PersonalNr + SeminarNr + Registration date *

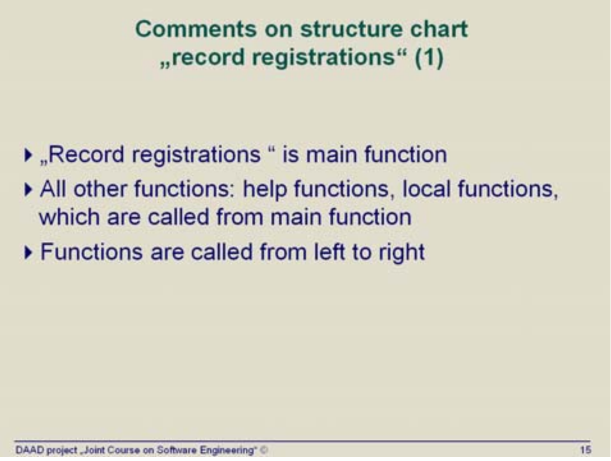
DAAD project „Joint Course on Software Engineering“ © 13



Structure chart: record registrations

Part of software architecture according to structured design

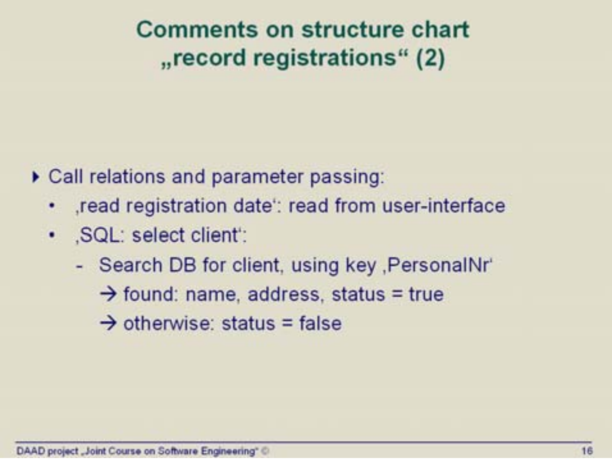
DAAD project „Joint Course on Software Engineering“ © 14



Comments on structure chart „record registrations“ (1)

- „Record registrations “ is main function
- All other functions: help functions, local functions, which are called from main function
- Functions are called from left to right

DAAD project „Joint Course on Software Engineering“ © 15



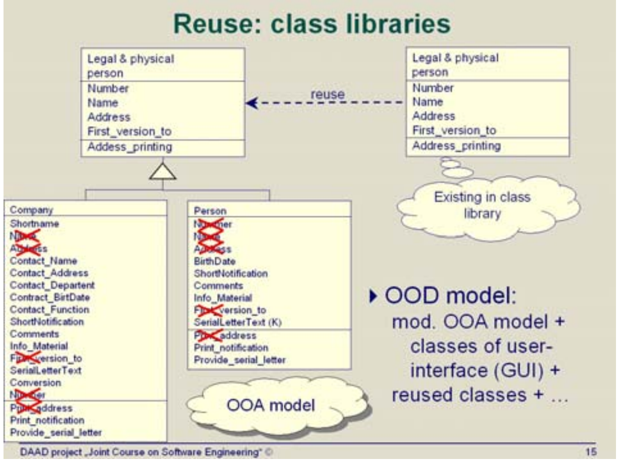
Comments on structure chart „record registrations“ (2)

- Call relations and parameter passing:
  - ‚read registration date‘: read from user-interface
  - ‚SQL: select client‘:
    - Search DB for client, using key ‚PersonalNr‘
      → found: name, address, status = true
      → otherwise: status = false

DAAD project „Joint Course on Software Engineering“ © 16

2.12 Topic 16: *Object-oriented design*

This topic uses just one slide describing a case study – the slide that elaborates on how class libraries can be used. However, this slide can remain even if the case study is changed, because it is general enough.



Reuse: class libraries

- OOD model: mod. OOA model + classes of user-interface (GUI) + reused classes + …

DAAD project „Joint Course on Software Engineering" © 15

2.13 Topics 25 and 26: *Introduction to software ergonomics* and *User manuals*

These two topics are not yet available in English.
Nevertheless:

To do:
- Implement a case study or provide a design of user interface.
- Write parts of user manual.

2.14 Summary

Activities in building a new case study are summarized as follows (this is only an approximation – for full description, you should nevertheless read the whole text). Activities are ordered by importance!

- Find a problem of reasonably large size an complexity (for example from textbooks, real projects or educational projects)
- Develop requirements specification
- Develop a full class diagram as the basis of object-oriented analysis
- Develop accompanying diagrams for the dynamic view of object-oriented analysis: state automata (object life cycle), activity diagrams, collaboration diagram, sequence diagram
- Develop a data-flow diagram for a significant part of requirements
- Perform the structured analysis of the system: develop a hierarchy of data flow diagrams for a significant part of the requirements
- Do a cost estimation
- Implement the case study
- Write parts of user manual

To replace the existing case study with the new one, one should replace about 120 slides in the lecture.

3 The supporting case-study – XCTL

XCTL is a realistic program used mainly in the lecture to illustrate a process of reverse engineering. To find out its basic characteristics a software measurements has been applied and also presented in the lecture.

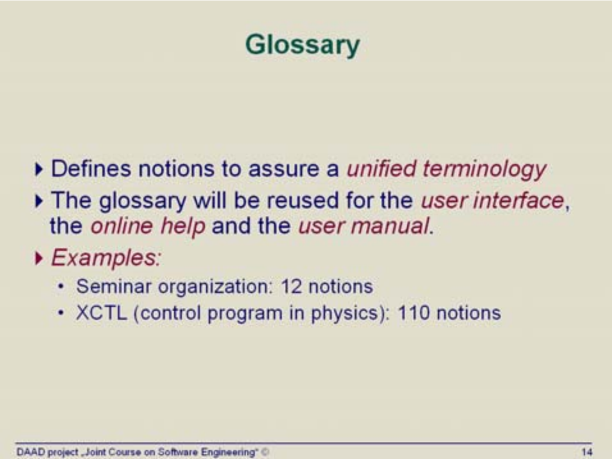
In the same way another case study could replace XCTL *or* illustrate other important aspects of software development.

3.1 Topic 5: *Results of the "Analysis and Definition" phase*

The supporting case study is for the first time mentioned in Topic 5: *Results of the "Analysis and Definition" phase*. It is used only in one slide to show how many notions are there in the glossary.

To do:
Develop at least glossary and use case diagram (as part of requirements specification) for a software product, that should be:
- of similar size as the current one (based on the number of use cases, for example),
- possibly already existing and of unknown structure (such that it is suitable for finding out the structure ☺)
- of such complexity:
    - that various software metrics methods can be applied

Option:
Develop a *full* requirements specification.



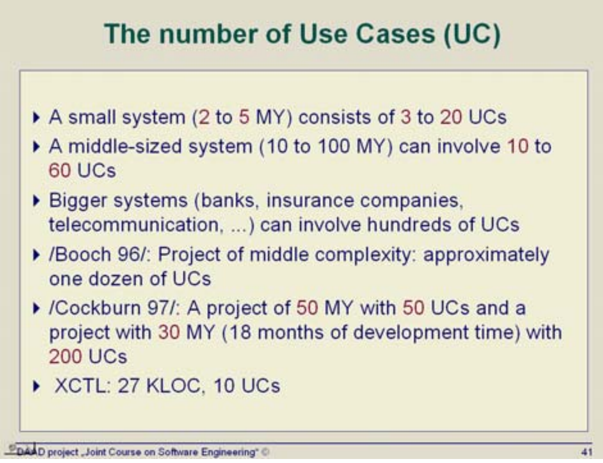
Glossary

- Defines notions to assure a *unified terminology*
- The glossary will be reused for the *user interface*, the *online help* and the *user manual*.
- *Examples:*
  - Seminar organization: 12 notions
  - XCTL (control program in physics): 110 notions

DAAD project „Joint Course on Software Engineering" © 14

3.2 Topic 7: *Function-oriented view*

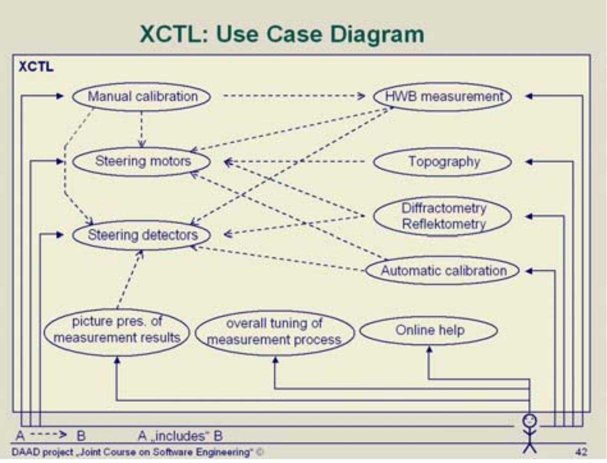
This topic uses the supporting case study to show its use cases.



The number of Use Cases (UC)

- A small system (2 to 5 MY) consists of 3 to 20 UCs
- A middle-sized system (10 to 100 MY) can involve 10 to 60 UCs
- Bigger systems (banks, insurance companies, telecommunication, ...) can involve hundreds of UCs
- /Booch 96/: Project of middle complexity: approximately one dozen of UCs
- /Cockburn 97/: A project of 50 MY with 50 UCs and a project with 30 MY (18 months of development time) with 200 UCs
- XCTL: 27 KLOC, 10 UCs

DAAD project „Joint Course on Software Engineering" © 41



XCTL: Use Case Diagram

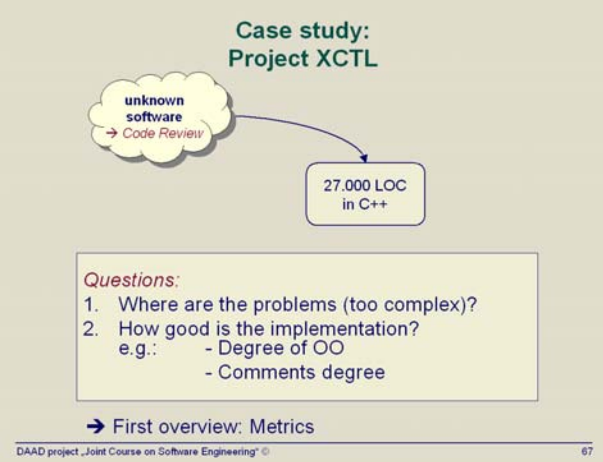
XCTL
Manual calibration
HWB measurement
Steering motors
Topography
Diffractometry Reflektometry
Steering detectors
Automatic calibration
picture pres. of measurement results
overall tuning of measurement process
Online help
A - - - -> B A „includes" B

DAAD project „Joint Course on Software Engineering" © 42

3.3 Topic 21: *Software metrics*

This topic uses the supporting case study to illustrate the application of several software metrics methods (see slides).

To do:
- Develop several characteristic measurements that can be used to illustrate major measurement techniques



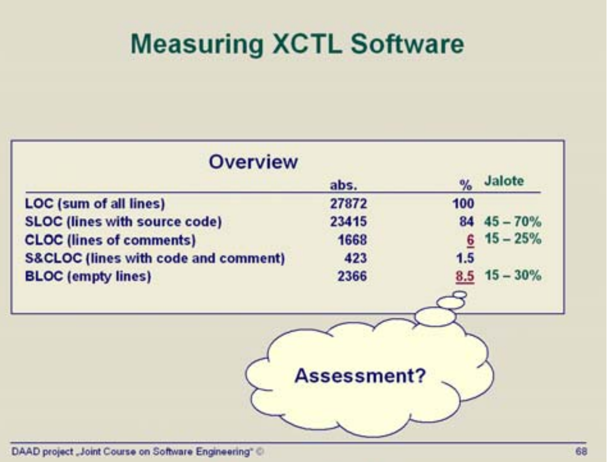
Case study: Project XCTL

unknown software
→ Code Review
27.000 LOC in C++

Questions:
1. Where are the problems (too complex)?
2. How good is the implementation?
   e.g.: - Degree of OO
   - Comments degree

→ First overview: Metrics

DAAD project „Joint Course on Software Engineering" © 67



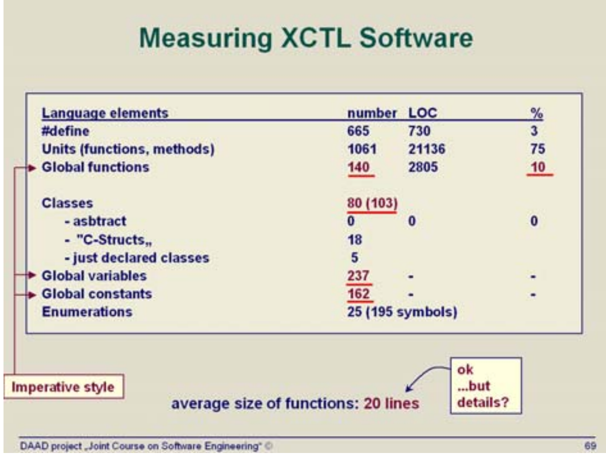
Measuring XCTL Software

Overview

	abs.	%	Jalote
LOC (sum of all lines)	27872	100	
SLOC (lines with source code)	23415	84	45 – 70%
CLOC (lines of comments)	1668	6	15 – 25%
S&CLOC (lines with code and comment)	423	1.5	
BLOC (empty lines)	2366	8.5	15 – 30%

Assessment?

DAAD project „Joint Course on Software Engineering" © 68



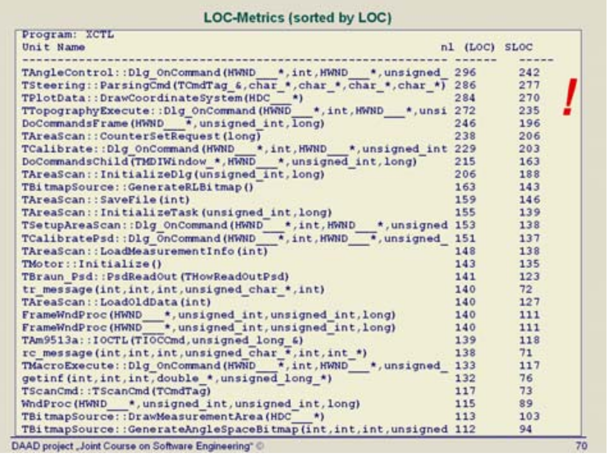
Measuring XCTL Software

Language elements	number	LOC	%
#define	665	730	3
Units (functions, methods)	1061	21136	75
Global functions	140	2805	10
Classes	80 (103)		
- asbtract	0	0	0
- "C-Structs„	18		
- just declared classes	5		
Global variables	237	-	-
Global constants	162	-	-
Enumerations	25 (195 symbols)		

Imperative style
ok
...but
details?

average size of functions: 20 lines

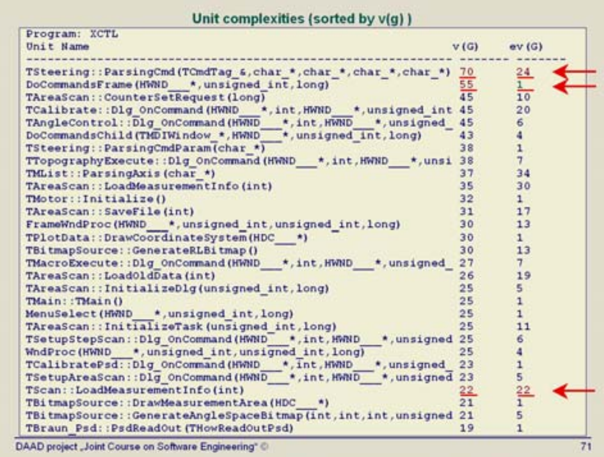
DAAD project „Joint Course on Software Engineering" © 69



LOC-Metrics (sorted by LOC)

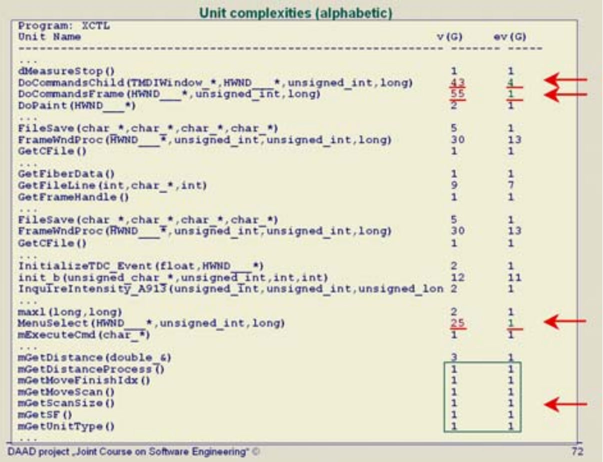
```
Program:  XCTL
Unit Name                                                  nl  (LOC)  SLOC
---------------------------------------------------------- ------   -----
TAngleControl::Dlg_OnCommand(HWND___*,int,HWND___*,unsigned_ 296      242
TSteering::ParsingCmd(TCmdTag_&,char_*,char_*,char_*,char_*) 286      277
TPlotData::DrawCoordinateSystem(HDC___*)                   284      270
TTopographyExecute::Dlg_OnCommand(HWND___*,int,HWND___*,unsi 272      235
DoCommandsFrame(HWND___*,unsigned_int,long)                246      196
TAreaScan::CounterSetRequest(long)                         238      206
TCalibrate::Dlg_OnCommand(HWND___*,int,HWND___*,unsigned_int 229      203
DoCommandsChild(TMDIWindow_*,HWND___*,unsigned_int,long)   215      163
TAreaScan::InitializeDlg(unsigned_int,long)                206      188
TBitmapSource::GenerateRLBitmap()                          163      143
TAreaScan::SaveFile(int)                                   159      146
TAreaScan::InitializeTask(unsigned_int,long)               155      139
TSetupAreaScan::Dlg_OnCommand(HWND___*,int,HWND___*,unsigned 153      138
TCalibratePsd::Dlg_OnCommand(HWND___*,int,HWND___*,unsigned_ 151      137
TAreaScan::LoadMeasurementInfo(int)                        148      138
TMotor::Initialize()                                       143      135
TBraun_Psd::PsdReadOut(THowReadOutPsd)                     141      123
tr_message(int,int,int,unsigned_char_*,int)                140      72
TAreaScan::LoadOldData(int)                                140      127
FrameWndProc(HWND___*,unsigned_int,unsigned_int,long)      140      111
FrameWndProc(HWND___*,unsigned_int,unsigned_int,long)      140      111
TAm9513a::IOCTL(TIOCCmd,unsigned_long_&)                   139      118
rc_message(int,int,int,unsigned_char_*,int,int_*)          138      71
TMacroExecute::Dlg_OnCommand(HWND___*,int,HWND___*,unsigned_ 133      117
getinf(int,int,int,double_*,unsigned_long_*)               132      76
TScanCmd::TScanCmd(TCmdTag)                                117      73
WndProc(HWND___*,unsigned_int,unsigned_int,long)           115      89
TBitmapSource::DrawMeasurementArea(HDC___*)                113      103
TBitmapSource::GenerateAngleSpaceBitmap(int,int,int,unsigned 112      94
```

DAAD project „Joint Course on Software Engineering" © 70



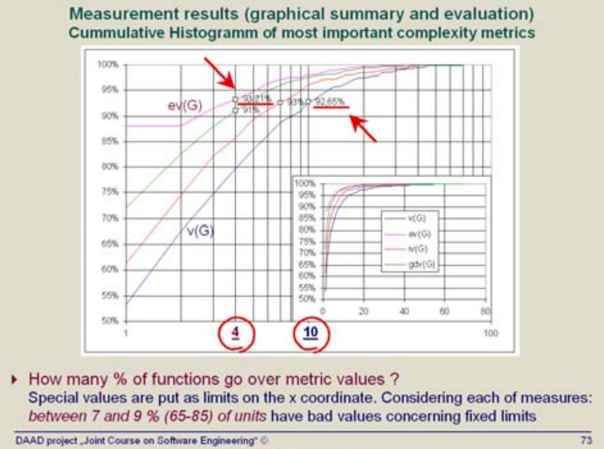
Unit complexities (sorted by v(g))

```
Program: XCTL
Unit Name                                                      v(G)    ev(G)
------------------------------------------------------------ ------- -------
TSteering::ParsingCmd(TCmdTag_&,char_*,char_*,char_*,char_*) 70      24
DoCommandsFrame(HWND___*,unsigned_int,long)                  55      1
TAreaScan::CounterSetRequest(long)                           45      10
TCalibrate::Dlg_OnCommand(HWND___*,int,HWND___*,unsigned_int 45      20
TAngleControl::Dlg_OnCommand(HWND___*,int,HWND___*,unsigned_ 45      6
DoCommandsChild(TMDIWindow_*,HWND___*,unsigned_int,long)     43      4
TSteering::ParsingCmdParam(char_*)                           38      1
TTopographyExecute::Dlg_OnCommand(HWND___*,int,HWND___*,unsi 38      7
TMList::ParsingAxis(char_*)                                  37      34
TAreaScan::LoadMeasurementInfo(int)                          35      30
TMotor::Initialize()                                         32      1
TAreaScan::SaveFile(int)                                     31      17
FrameWndProc(HWND___*,unsigned_int,unsigned_int,long)        30      13
TPlotData::DrawCoordinateSystem(HDC___*)                     30      1
TBitmapSource::GenerateRLBitmap()                            30      13
TMacroExecute::Dlg_OnCommand(HWND___*,int,HWND___*,unsigned_ 27      7
TAreaScan::LoadOldData(int)                                  26      19
TAreaScan::InitializeDlg(unsigned_int,long)                  25      5
TMain::TMain()                                               25      1
MenuSelect(HWND___*,unsigned_int,long)                       25      1
TAreaScan::InitializeTask(unsigned_int,long)                 25      11
TSetupStepScan::Dlg_OnCommand(HWND___*,int,HWND___*,unsigned 25      6
WndProc(HWND___*,unsigned_int,unsigned_int,long)             25      4
TCalibratePsd::Dlg_OnCommand(HWND___*,int,HWND___*,unsigned_ 23      1
TSetupAreaScan::Dlg_OnCommand(HWND___*,int,HWND___*,unsigned 23      5
TScan::LoadMeasurementInfo(int)                              22      22
TBitmapSource::DrawMeasurementArea(HDC___*)                  21      1
TBitmapSource::GenerateAngleSpaceBitmap(int,int,int,unsigned 21      5
TBraun_Psd::PsdReadOut(THowReadOutPsd)                       19      1
```



Unit complexities (alphabetic)

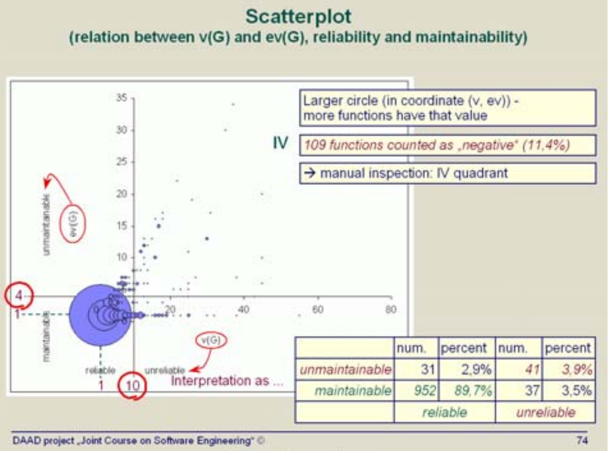
```
Program: XCTL
Unit Name                                                       v(G)    ev(G)
-------------------------------------------------------------- ------- -----
...
dMeasureStop()                                                  1       1
DoCommandsChild(TMDIWindow_*,HWND___*,unsigned_int,long)        43      4
DoCommandsFrame(HWND___*,unsigned_int,long)                     55      1
DoPaint(HWND___*)                                               2       1
...
FileSave(char_*,char_*,char_*,char_*)                           5       1
FrameWndProc(HWND___*,unsigned_int,unsigned_int,long)           30      13
GetCFile()                                                      1       1
...
GetFiberData()                                                  1       1
GetFileLine(int,char_*,int)                                     9       7
GetFrameHandle()                                                1       1
...
FileSave(char_*,char_*,char_*,char_*)                           5       1
FrameWndProc(HWND___*,unsigned_int,unsigned_int,long)           30      13
GetCFile()                                                      1       1
...
InitializeTDC_Event(float,HWND___*)                             2       1
init_b(unsigned_char_*,unsigned_int,int,int)                    12      11
InquireIntensity_A913(unsigned_int,unsigned_int,unsigned_lon    2       1
...
maxl(long,long)                                                 2       1
MenuSelect(HWND___*,unsigned_int,long)                          25      1
mExecuteCmd(char_*)                                             1       1
...
mGetDistance(double_&)                                          3       1
mGetDistanceProcess()                                           1       1
mGetMoveFinishIdx()                                             1       1
mGetMoveScan()                                                  1       1
mGetScanSize()                                                  1       1
mGetSF()                                                        1       1
mGetUnitType()                                                  1       1
...
```



Measurement results (graphical summary and evaluation)

Cummulative Histogramm of most important complexity metrics

- How many % of functions go over metric values ?
  Special values are put as limits on the x coordinate. Considering each of measures: *between 7 and 9 % (65-85) of units* have bad values concerning fixed limits



Scatterplot

(relation between v(G) and ev(G), reliability and maintainability)

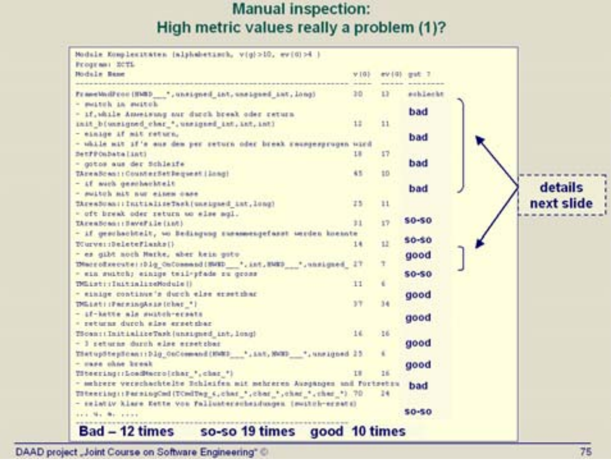
Larger circle (in coordinate (v, ev)) - more functions have that value

109 functions counted as „negative“ (11,4%)

→ manual inspection: IV quadrant

Interpretation as ...

	num.	percent	num.	percent
unmaintainable	31	2,9%	*41*	*3,9%*
maintainable	*952*	*89,7%*	37	3,5%
	reliable		*unreliable*	



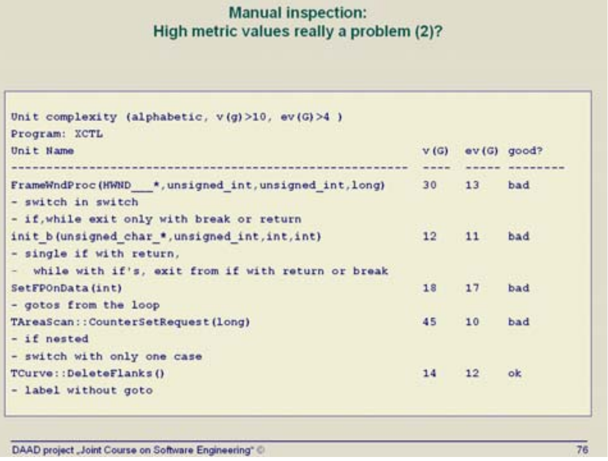
Manual inspection:
High metric values really a problem (1)?

```
Module Komplexitaten (alphabetisch, v(g)>10, ev(G)>4 )
Program: XCTL
Module Name                                                  v(G)  ev(G) gut ?
------------------------------------------------------------ ----  ----- --------
FrameWndProc(HWND___*,unsigned_int,unsigned_int,long)        30    13    schlecht
- switch in switch
- if,while Anweisung nur durch break oder return
init_b(unsigned_char_*,unsigned_int,int,int)                 12    11
- einige if mit return,
- while mit if's aus dem per return oder break rausgesprugen wird
SetFPOnData(int)                                             18    17
- gotos aus der Schleife
TAreaScan::CounterSetRequest(long)                           45    10
- if auch geschachtelt
- switch mit nur einem case
TAreaScan::InitializeTask(unsigned_int,long)                 25    11
- oft break oder return wo else mgl.
TAreaScan::SaveFile(int)                                     31    17
- if geschachtelt, wo Bedingung zusammengefasst werden koennte
TCurve::DeleteFlanks()                                       14    12
- es gibt noch Marke, aber kein goto
TMacroExecute::Dlg_OnCommand(HWND___*,int,HWND___*,unsigned_ 27    7
- ein switch; einige teil-pfade zu gross
TMList::InitializeModule()                                   11    6
- einige continue's durch else ersetzbar
TMList::ParsingAxis(char_*)                                  37    34
- if-kette als switch-ersatz
- returns durch else ersetzbar
TScan::InitializeTask(unsigned_int,long)                     16    16
- 3 returns durch else ersetzbar
TSetupStepScan::Dlg_OnCommand(HWND___*,int,HWND___*,unsigned 25    6
- case ohne break
TSteering::LoadMacro(char_*,char_*)                          18    16
- mehrere verschachtelte Schleifen mit mehreren Ausgangen und Fortsetzu
TSteering::ParsingCmd(TCmdTag_&,char_*,char_*,char_*,char_*) 70    24
- relativ klare Kette von Fallunterscheidungen (switch-ersatz)
... %. %. ....
------------------------------------------------------------
```

bad
bad
bad
bad
so-so
so-so
good
so-so
good
good
good
good
bad
so-so

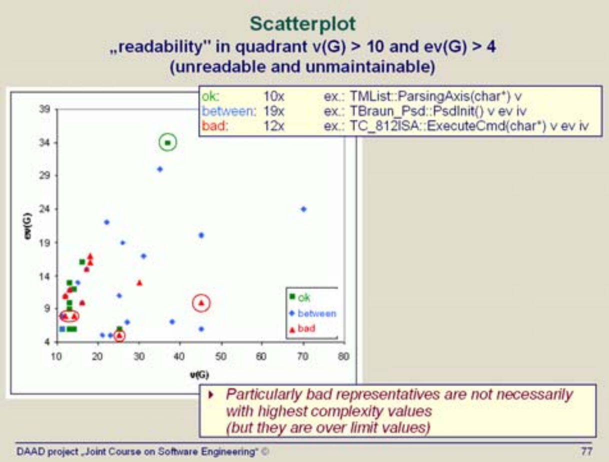
details next slide

Bad – 12 times so-so 19 times good 10 times



Manual inspection:
High metric values really a problem (2)?

```
Unit complexity (alphabetic, v(g)>10, ev(G)>4 )
Program: XCTL
Unit Name                                                  v(G)  ev(G)  good?
---------------------------------------------------------  ----  -----  --------
FrameWndProc(HWND___*,unsigned_int,unsigned_int,long)      30    13     bad
- switch in switch
- if,while exit only with break or return
init_b(unsigned_char_*,unsigned_int,int,int)               12    11     bad
- single if with return,
-  while with if's, exit from if with return or break
SetFPOnData(int)                                           18    17     bad
- gotos from the loop
TAreaScan::CounterSetRequest(long)                         45    10     bad
- if nested
- switch with only one case
TCurve::DeleteFlanks()                                     14    12     ok
- label without goto
```

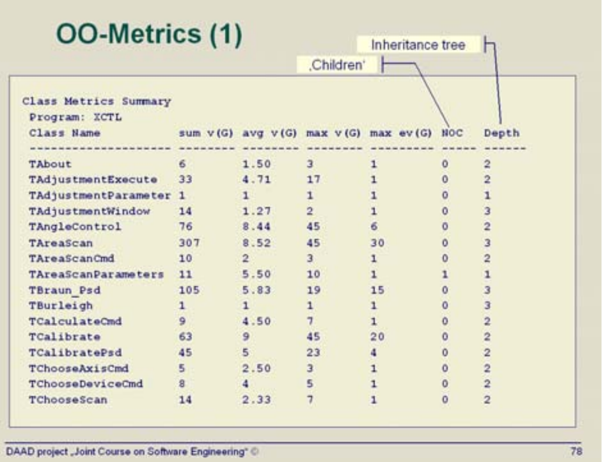


Scatterplot

„readability" in quadrant v(G) > 10 and ev(G) > 4
(unreadable and unmaintainable)

ok:	10x	ex.: TMList::ParsingAxis(char*) v
between:	19x	ex.: TBraun_Psd::PsdInit() v ev iv
bad:	12x	ex.: TC_812ISA::ExecuteCmd(char*) v ev iv

- *Particularly bad representatives are not necessarily with highest complexity values (but they are over limit values)*

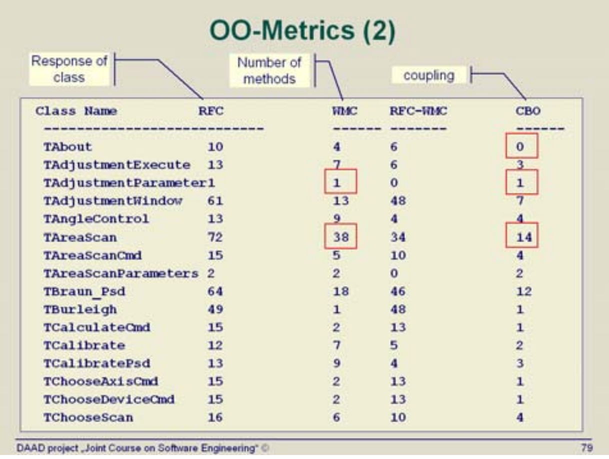


OO-Metrics (1)

Inheritance tree

‚Children'

```
Class Metrics Summary
 Program: XCTL
 Class Name           sum v(G) avg v(G) max v(G) max ev(G) NOC   Depth
 -------------------- -------- -------- -------- --------- ----- ------
 TAbout               6        1.50     3        1         0     2
 TAdjustmentExecute   33       4.71     17       1         0     2
 TAdjustmentParameter 1        1        1        1         0     1
 TAdjustmentWindow    14       1.27     2        1         0     3
 TAngleControl        76       8.44     45       6         0     2
 TAreaScan            307      8.52     45       30        0     3
 TAreaScanCmd         10       2        3        1         0     2
 TAreaScanParameters  11       5.50     10       1         1     1
 TBraun_Psd           105      5.83     19       15        0     3
 TBurleigh            1        1        1        1         0     3
 TCalculateCmd        9        4.50     7        1         0     2
 TCalibrate           63       9        45       20        0     2
 TCalibratePsd        45       5        23       4         0     2
 TChooseAxisCmd       5        2.50     3        1         0     2
 TChooseDeviceCmd     8        4        5        1         0     2
 TChooseScan          14       2.33     7        1         0     2
```



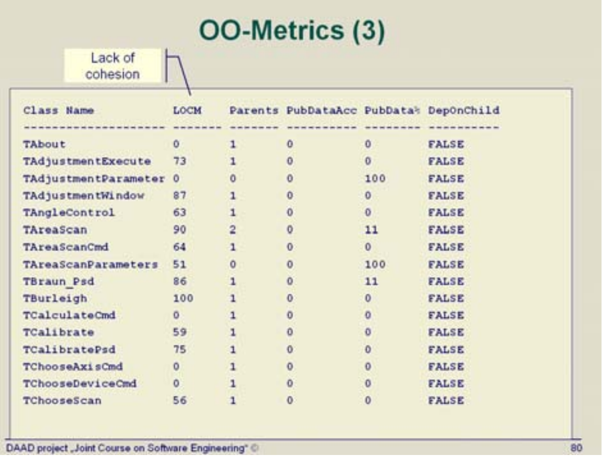
OO-Metrics (2)

Response of class

Number of methods

coupling

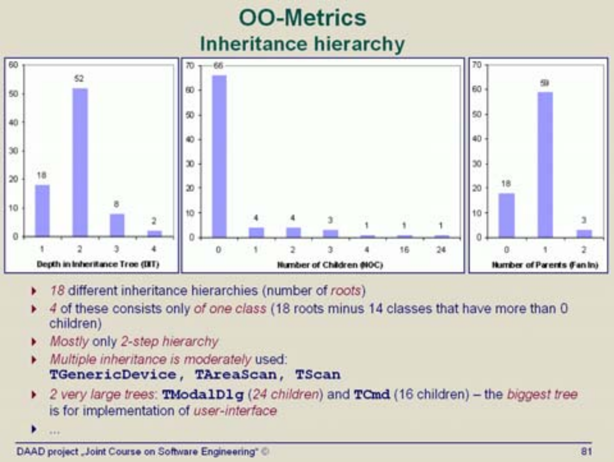
```
Class Name           RFC            WMC    RFC-WMC      CBO
 ---------------------------        ------ -------      ------
 TAbout               10            4      6            0
 TAdjustmentExecute   13            7      6            3
 TAdjustmentParameter1              1      0            1
 TAdjustmentWindow    61            13     48           7
 TAngleControl        13            9      4            4
 TAreaScan            72            38     34           14
 TAreaScanCmd         15            5      10           4
 TAreaScanParameters  2             2      0            2
 TBraun_Psd           64            18     46           12
 TBurleigh            49            1      48           1
 TCalculateCmd        15            2      13           1
 TCalibrate           12            7      5            2
 TCalibratePsd        13            9      4            3
 TChooseAxisCmd       15            2      13           1
 TChooseDeviceCmd     15            2      13           1
 TChooseScan          16            6      10           4
```



OO-Metrics (3)

Lack of cohesion

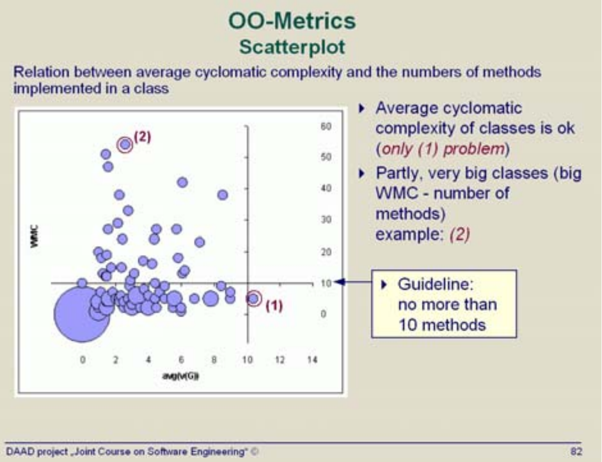
```
Class Name           LOCM    Parents PubDataAcc PubData% DepOnChild
-------------------- ------- ------- ---------- -------- ----------
TAbout               0       1       0          0        FALSE
TAdjustmentExecute   73      1       0          0        FALSE
TAdjustmentParameter 0       0       0          100      FALSE
TAdjustmentWindow    87      1       0          0        FALSE
TAngleControl        63      1       0          0        FALSE
TAreaScan            90      2       0          11       FALSE
TAreaScanCmd         64      1       0          0        FALSE
TAreaScanParameters  51      0       0          100      FALSE
TBraun_Psd           86      1       0          11       FALSE
TBurleigh            100     1       0          0        FALSE
TCalculateCmd        0       1       0          0        FALSE
TCalibrate           59      1       0          0        FALSE
TCalibratePsd        75      1       0          0        FALSE
TChooseAxisCmd       0       1       0          0        FALSE
TChooseDeviceCmd     0       1       0          0        FALSE
TChooseScan          56      1       0          0        FALSE
```



OO-Metrics
Inheritance hierarchy

- *18* different inheritance hierarchies (number of *roots*)
- *4* of these consists only *of one class* (18 roots minus 14 classes that have more than 0 children)
- *Mostly* only *2-step hierarchy*
- *Multiple inheritance is moderately* used: **TGenericDevice, TAreaScan, TScan**
- *2 very large trees*: **TModalDlg** (*24 children*) and **TCmd** (16 children) – the *biggest tree* is for implementation of *user-interface*
- ...

DAAD project „Joint Course on Software Engineering“ © 81



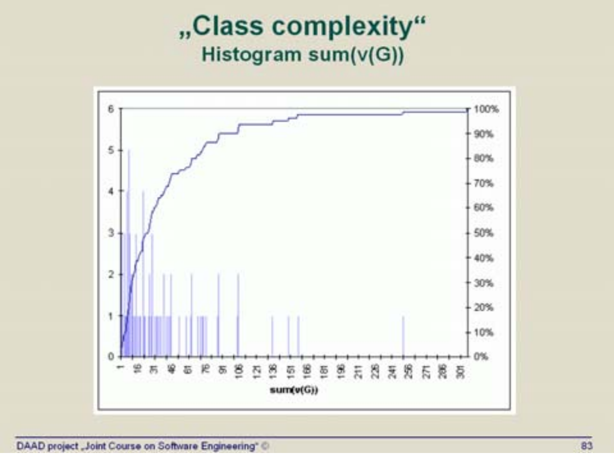
OO-Metrics
Scatterplot

Relation between average cyclomatic complexity and the numbers of methods implemented in a class

- Average cyclomatic complexity of classes is ok (*only (1) problem*)
- Partly, very big classes (big WMC - number of methods) example: *(2)*

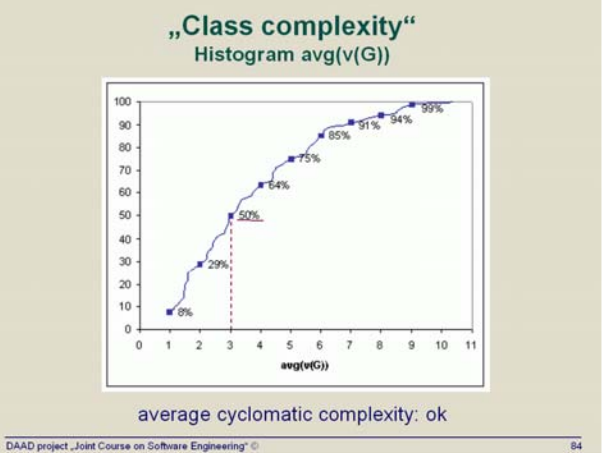
- Guideline: no more than 10 methods

DAAD project „Joint Course on Software Engineering“ © 82



„Class complexity“
Histogram sum(v(G))

DAAD project „Joint Course on Software Engineering“ © 83



„Class complexity“
Histogram avg(v(G))

average cyclomatic complexity: ok

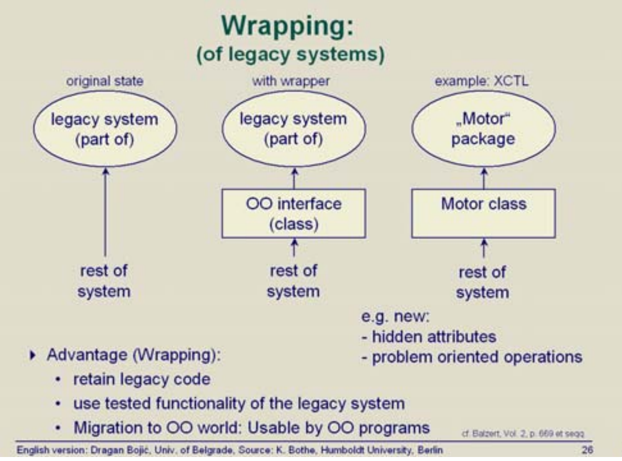
DAAD project „Joint Course on Software Engineering“ © 84

3.4 Topic 23: *Reverse engineering*

This topic uses supporting case study to show a realistic process of a reverse engineering.

To do:
- Develop (or simulate) a similar process to be shown to students.

Slides are used in two ways in the lecture.
<u>*Slides illustrating wrapping and giving overview of used tools*</u>.



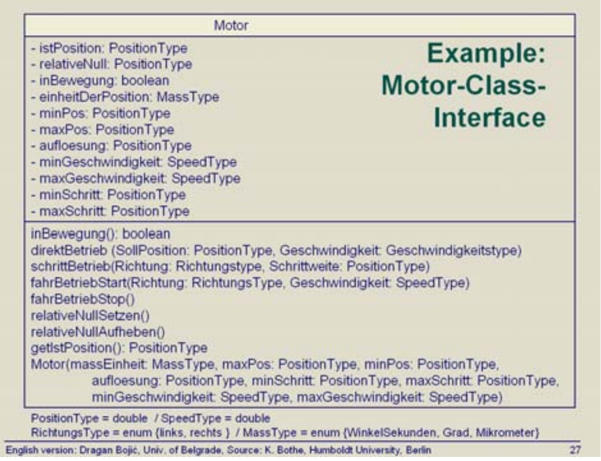
Wrapping:
(of legacy systems)

e.g. new:
- hidden attributes
- problem oriented operations

- Advantage (Wrapping):
  - retain legacy code
  - use tested functionality of the legacy system
  - Migration to OO world: Usable by OO programs

cf. Balzert, Vol. 2, p. 669 et seqq.

English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 26

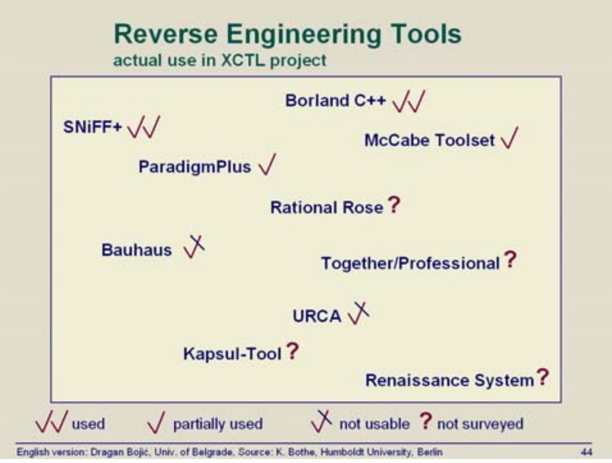


Example: Motor-Class-Interface

```
Motor
- istPosition: PositionType
- relativeNull: PositionType
- inBewegung: boolean
- einheitDerPosition: MassType
- minPos: PositionType
- maxPos: PositionType
- aufloesung: PositionType
- minGeschwindigkeit: SpeedType
- maxGeschwindigkeit: SpeedType
- minSchritt: PositionType
- maxSchritt: PositionType
inBewegung(): boolean
direktBetrieb (SollPosition: PositionType, Geschwindigkeit: Geschwindigkeitstype)
schrittBetrieb(Richtung: Richtungstype, Schrittweite: PositionType)
fahrBetriebStart(Richtung: RichtungsType, Geschwindigkeit: SpeedType)
fahrBetriebStop()
relativeNullSetzen()
relativeNullAufheben()
getIstPosition(): PositionType
Motor(massEinheit: MassType, maxPos: PositionType, minPos: PositionType,
        aufloesung: PositionType, minSchritt: PositionType, maxSchritt: PositionType,
        minGeschwindigkeit: SpeedType, maxGeschwindigkeit: SpeedType)
```

PositionType = double / SpeedType = double
RichtungsType = enum {links, rechts } / MassType = enum {WinkelSekunden, Grad, Mikrometer}

English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 27



Reverse Engineering Tools

actual use in XCTL project

Borland C++ ✓✓

SNiFF+ ✓✓

McCabe Toolset ✓

ParadigmPlus ✓

Rational Rose ?

Bauhaus ✗

Together/Professional ?

URCA ✗

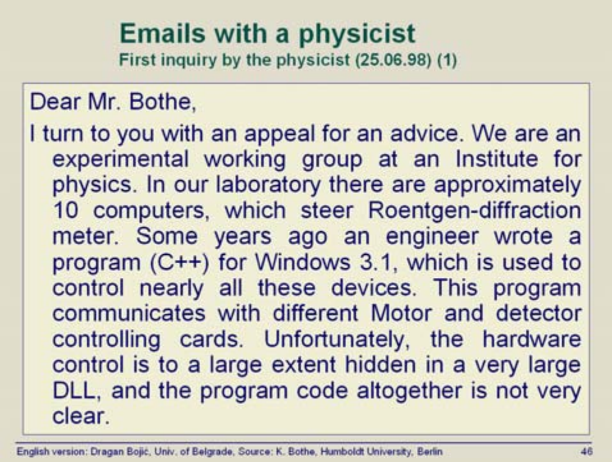
Kapsul-Tool ?

Renaissance System ?

✓✓ used ✓ partially used ✗ not usable ? not surveyed

English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 44

Slides illustrating the whole process of reverse engineering, incl. history.



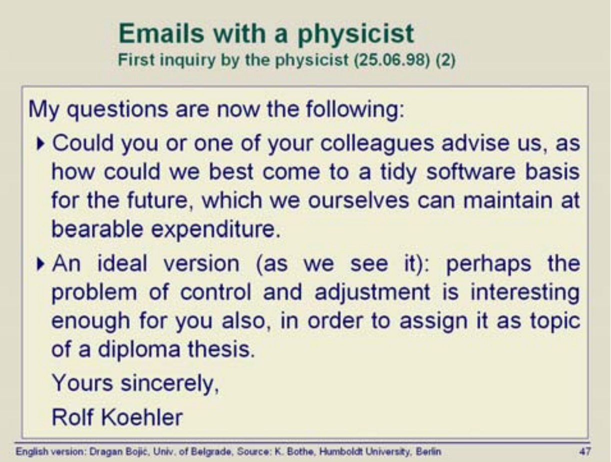
Emails with a physicist

First inquiry by the physicist (25.06.98) (1)

Dear Mr. Bothe,

I turn to you with an appeal for an advice. We are an experimental working group at an Institute for physics. In our laboratory there are approximately 10 computers, which steer Roentgen-diffraction meter. Some years ago an engineer wrote a program (C++) for Windows 3.1, which is used to control nearly all these devices. This program communicates with different Motor and detector controlling cards. Unfortunately, the hardware control is to a large extent hidden in a very large DLL, and the program code altogether is not very clear.

English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 46



Emails with a physicist

First inquiry by the physicist (25.06.98) (2)

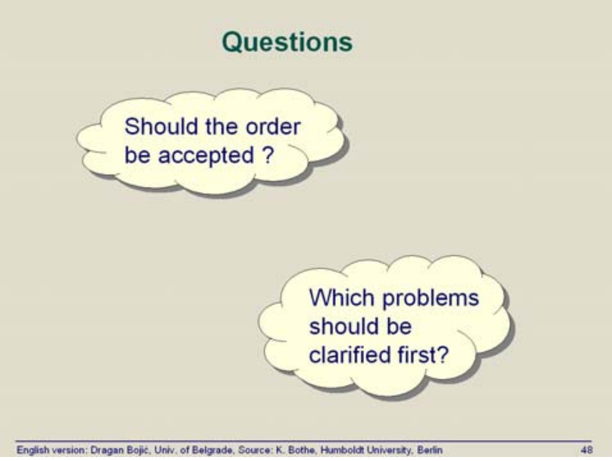
My questions are now the following:

- Could you or one of your colleagues advise us, as how could we best come to a tidy software basis for the future, which we ourselves can maintain at bearable expenditure.
- An ideal version (as we see it): perhaps the problem of control and adjustment is interesting enough for you also, in order to assign it as topic of a diploma thesis.

Yours sincerely,

Rolf Koehler

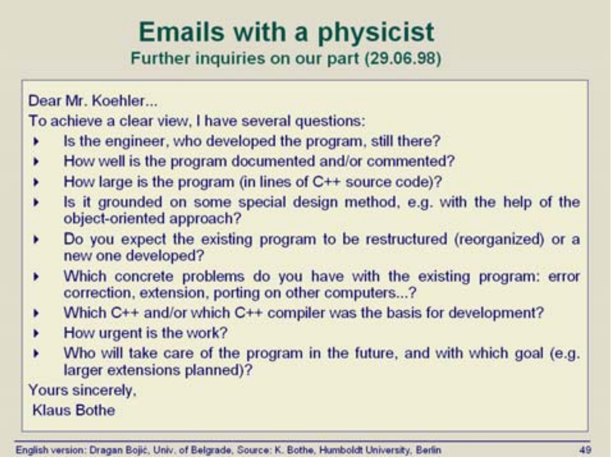
English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 47



Questions

Should the order be accepted ?
Which problems should be clarified first?

English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 48



Emails with a physicist

Further inquiries on our part (29.06.98)

Dear Mr. Koehler...

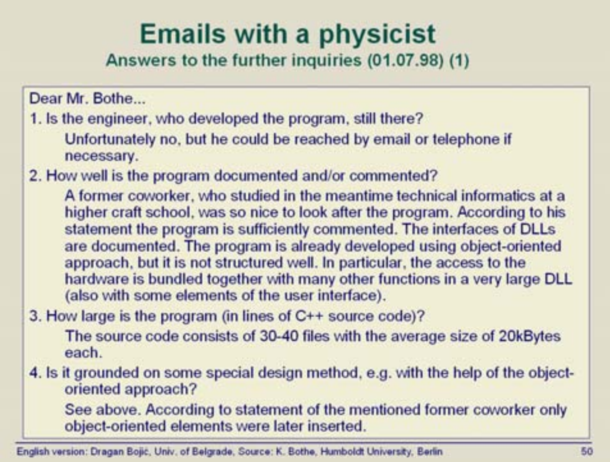
To achieve a clear view, I have several questions:

- Is the engineer, who developed the program, still there?
- How well is the program documented and/or commented?
- How large is the program (in lines of C++ source code)?
- Is it grounded on some special design method, e.g. with the help of the object-oriented approach?
- Do you expect the existing program to be restructured (reorganized) or a new one developed?
- Which concrete problems do you have with the existing program: error correction, extension, porting on other computers...?
- Which C++ and/or which C++ compiler was the basis for development?
- How urgent is the work?
- Who will take care of the program in the future, and with which goal (e.g. larger extensions planned)?

Yours sincerely,

Klaus Bothe

English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 49



Emails with a physicist

Answers to the further inquiries (01.07.98) (1)

Dear Mr. Bothe...

1. Is the engineer, who developed the program, still there?

   Unfortunately no, but he could be reached by email or telephone if necessary.

2. How well is the program documented and/or commented?

   A former coworker, who studied in the meantime technical informatics at a higher craft school, was so nice to look after the program. According to his statement the program is sufficiently commented. The interfaces of DLLs are documented. The program is already developed using object-oriented approach, but it is not structured well. In particular, the access to the hardware is bundled together with many other functions in a very large DLL (also with some elements of the user interface).

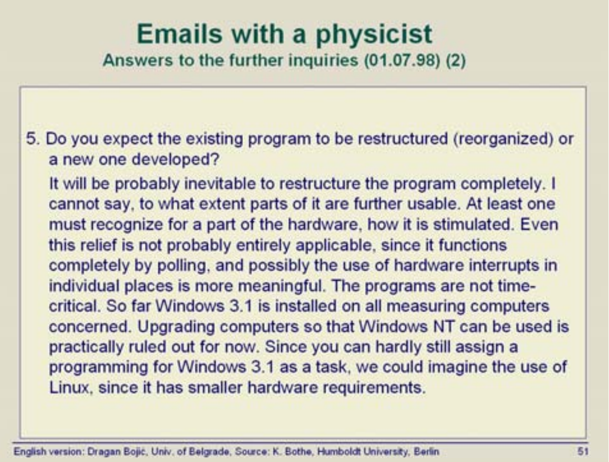
3. How large is the program (in lines of C++ source code)?

   The source code consists of 30-40 files with the average size of 20kBytes each.

4. Is it grounded on some special design method, e.g. with the help of the object-oriented approach?

   See above. According to statement of the mentioned former coworker only object-oriented elements were later inserted.

English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 50



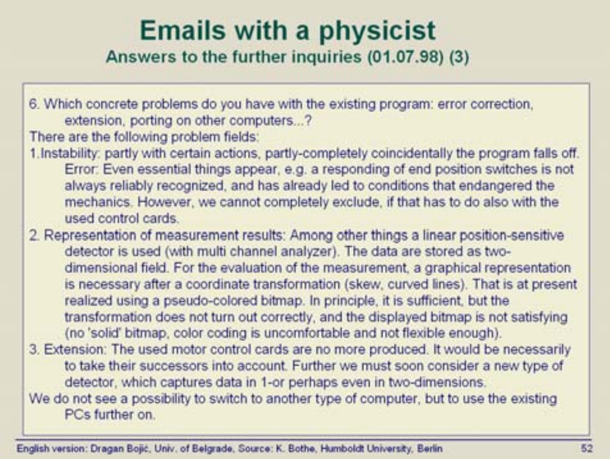
Emails with a physicist

Answers to the further inquiries (01.07.98) (2)

5. Do you expect the existing program to be restructured (reorganized) or a new one developed?

   It will be probably inevitable to restructure the program completely. I cannot say, to what extent parts of it are further usable. At least one must recognize for a part of the hardware, how it is stimulated. Even this relief is not probably entirely applicable, since it functions completely by polling, and possibly the use of hardware interrupts in individual places is more meaningful. The programs are not time-critical. So far Windows 3.1 is installed on all measuring computers concerned. Upgrading computers so that Windows NT can be used is practically ruled out for now. Since you can hardly still assign a programming for Windows 3.1 as a task, we could imagine the use of Linux, since it has smaller hardware requirements.

English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 51



Emails with a physicist

Answers to the further inquiries (01.07.98) (3)

6. Which concrete problems do you have with the existing program: error correction, extension, porting on other computers...?

There are the following problem fields:

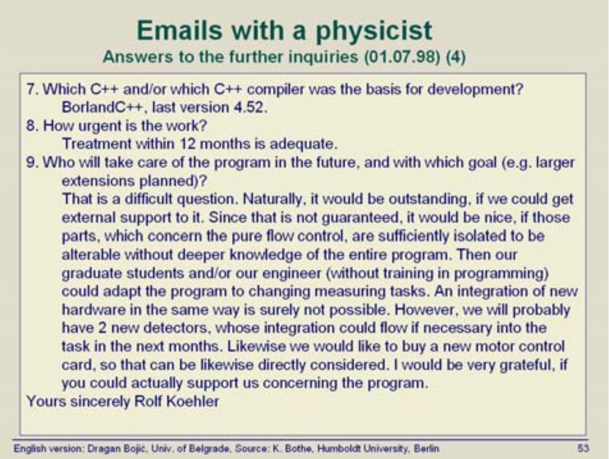
1.Instability: partly with certain actions, partly-completely coincidentally the program falls off. Error: Even essential things appear, e.g. a responding of end position switches is not always reliably recognized, and has already led to conditions that endangered the mechanics. However, we cannot completely exclude, if that has to do also with the used control cards.

2. Representation of measurement results: Among other things a linear position-sensitive detector is used (with multi channel analyzer). The data are stored as two-dimensional field. For the evaluation of the measurement, a graphical representation is necessary after a coordinate transformation (skew, curved lines). That is at present realized using a pseudo-colored bitmap. In principle, it is sufficient, but the transformation does not turn out correctly, and the displayed bitmap is not satisfying (no 'solid' bitmap, color coding is uncomfortable and not flexible enough).

3. Extension: The used motor control cards are no more produced. It would be necessarily to take their successors into account. Further we must soon consider a new type of detector, which captures data in 1-or perhaps even in two-dimensions.

We do not see a possibility to switch to another type of computer, but to use the existing PCs further on.

English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 52



Emails with a physicist

Answers to the further inquiries (01.07.98) (4)

7. Which C++ and/or which C++ compiler was the basis for development?

   BorlandC++, last version 4.52.

8. How urgent is the work?

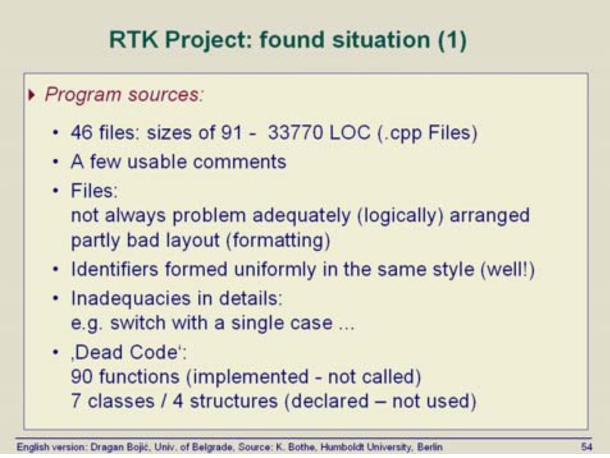
   Treatment within 12 months is adequate.

9. Who will take care of the program in the future, and with which goal (e.g. larger extensions planned)?

   That is a difficult question. Naturally, it would be outstanding, if we could get external support to it. Since that is not guaranteed, it would be nice, if those parts, which concern the pure flow control, are sufficiently isolated to be alterable without deeper knowledge of the entire program. Then our graduate students and/or our engineer (without training in programming) could adapt the program to changing measuring tasks. An integration of new hardware in the same way is surely not possible. However, we will probably have 2 new detectors, whose integration could flow if necessary into the task in the next months. Likewise we would like to buy a new motor control card, so that can be likewise directly considered. I would be very grateful, if you could actually support us concerning the program.

Yours sincerely Rolf Koehler

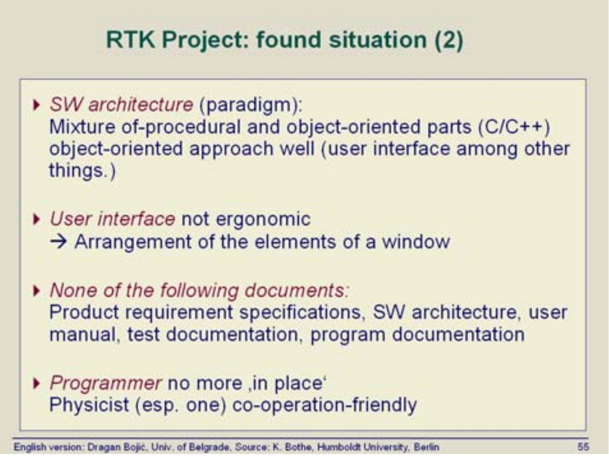
English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 53



RTK Project: found situation (1)

- *Program sources:*
  - 46 files: sizes of 91 - 33770 LOC (.cpp Files)
  - A few usable comments
  - Files:
    not always problem adequately (logically) arranged
    partly bad layout (formatting)
  - Identifiers formed uniformly in the same style (well!)
  - Inadequacies in details:
    e.g. switch with a single case ...
  - ‚Dead Code‘:
    90 functions (implemented - not called)
    7 classes / 4 structures (declared – not used)

English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 54



RTK Project: found situation (2)

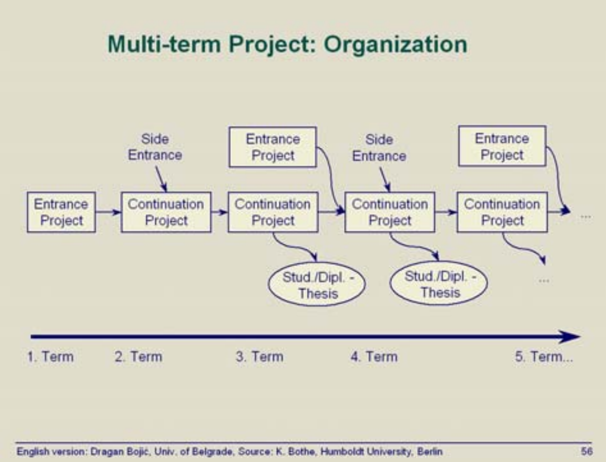
- *SW architecture* (paradigm):
  Mixture of-procedural and object-oriented parts (C/C++)
  object-oriented approach well (user interface among other things.)

- *User interface* not ergonomic
  → Arrangement of the elements of a window

- *None of the following documents:*
  Product requirement specifications, SW architecture, user manual, test documentation, program documentation

- *Programmer* no more ‚in place‘
  Physicist (esp. one) co-operation-friendly

English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 55



Multi-term Project: Organization

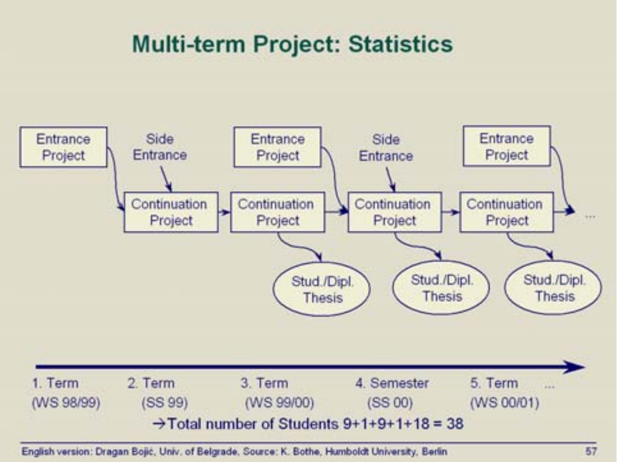
Side Entrance
Entrance Project
Side Entrance
Entrance Project

Entrance Project → Continuation Project → Continuation Project → Continuation Project → Continuation Project → ...

Stud./Dipl. - Thesis
Stud./Dipl. - Thesis
...

1. Term
2. Term
3. Term
4. Term
5. Term...

English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 56



Multi-term Project: Statistics

Entrance Project
Side Entrance
Entrance Project
Side Entrance
Entrance Project

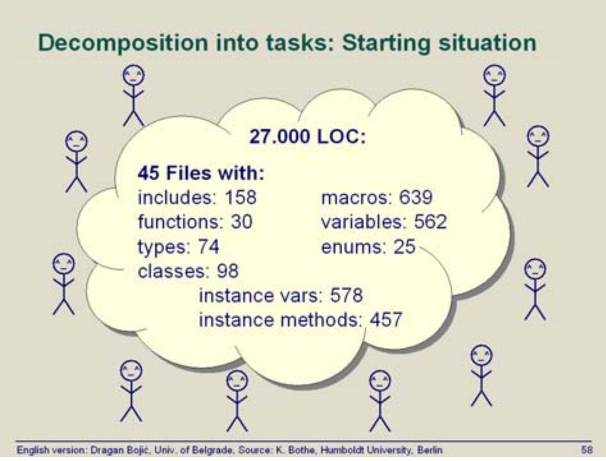
Continuation Project → Continuation Project → Continuation Project → Continuation Project → ...

Stud./Dipl. Thesis
Stud./Dipl. Thesis
Stud./Dipl. Thesis

1. Term (WS 98/99)
2. Term (SS 99)
3. Term (WS 99/00)
4. Semester (SS 00)
5. Term (WS 00/01) ...

→Total number of Students 9+1+9+1+18 = 38

English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 57



Decomposition into tasks: Starting situation

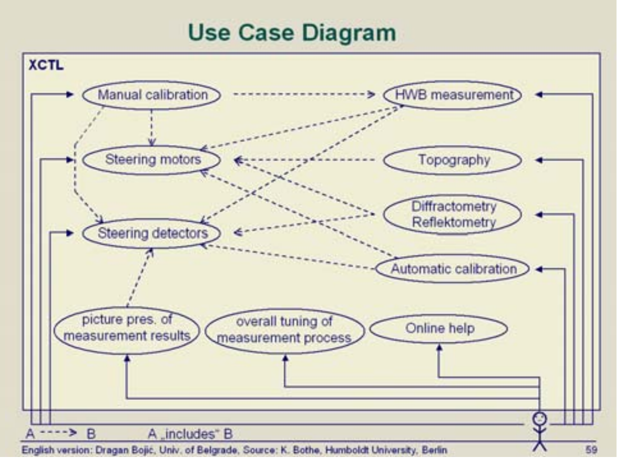
27.000 LOC:

45 Files with:

includes: 158
functions: 30
types: 74
classes: 98
instance vars: 578
instance methods: 457

macros: 639
variables: 562
enums: 25

English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 58



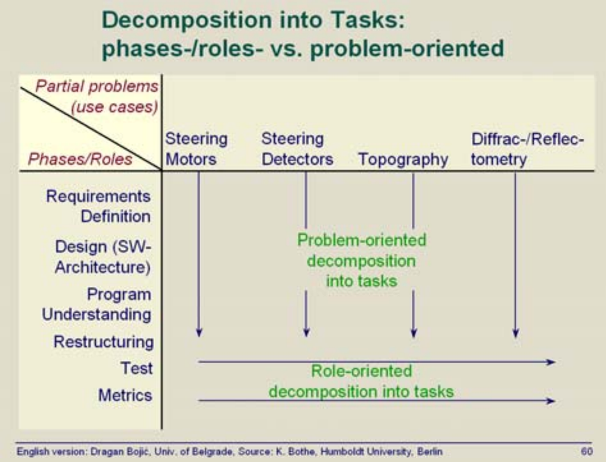
Use Case Diagram

XCTL

Manual calibration
HWB measurement
Steering motors
Topography
Diffractometry Reflektometry
Steering detectors
Automatic calibration
picture pres. of measurement results
overall tuning of measurement process
Online help

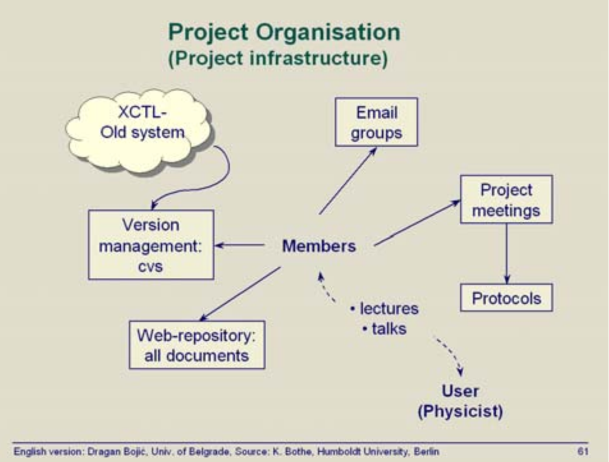
A ----> B A „includes" B

English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 59



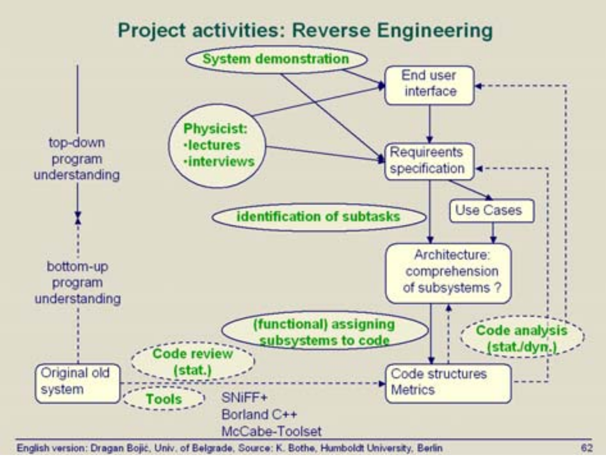
Decomposition into Tasks: phases-/roles- vs. problem-oriented

English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 60



Project Organisation (Project infrastructure)

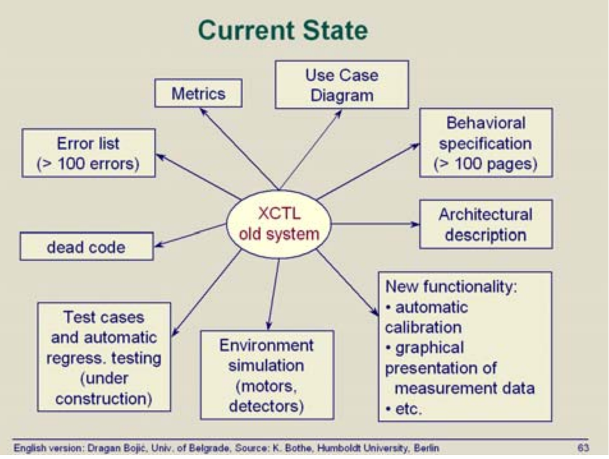
English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 61



Project activities: Reverse Engineering

System demonstration
End user interface
top-down program understanding
Physicist: ·lectures ·interviews
Requireents specification
Use Cases
identification of subtasks
Architecture: comprehension of subsystems ?
bottom-up program understanding
(functional) assigning subsystems to code
Code analysis (stat./dyn.)
Code review (stat.)
Original old system
Tools
SNiFF+
Borland C++
McCabe-Toolset
Code structures Metrics

English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 62



Current State

Metrics
Use Case Diagram
Behavioral specification (> 100 pages)
Error list (> 100 errors)
XCTL old system
Architectural description
dead code
Test cases and automatic regress. testing (under construction)
Environment simulation (motors, detectors)
New functionality: • automatic calibration • graphical presentation of measurement data • etc.

English version: Dragan Bojić, Univ. of Belgrade, Source: K. Bothe, Humboldt University, Berlin 63

3.5 Summary

Activities in building a new supporting case study should be summarized as follows (this is only an approximation – for full description, you should nevertheless read the whole text).

To replace the existing case study with the new one, one should replace about 33 slides in the lecture.

A supporting case study can be developed either to replace XCTL (mainly in topics on software metrics and reverse engineering) *or* to illustrate other important aspects of software development.

4 Possible further extensions

This section lists possibilities to further include the main case study into the lecture's topics.

Topics	**Currently**	**Possibilities to include also in**
1. What is software engineering		
2. Quality criteria for software products		
3. Software process models		
4. Basic concepts for software development documents		
5. Results of the "analysis and definition" phase	√	
6. Cost estimation	√	
7. Function-oriented view	√	
8. Data-oriented view	√	
9. Rule-oriented view	√	
10. Structured analysis	√	
11. State-oriented view	√	
12. Scenario-oriented view	√	
13. Object-oriented analysis	√	
14. Formal software specification and program verification		
15. Overview of design activities		
16. Structured design	√	
17. Object-oriented design	√	
18. Implementation		√
19. Systematic testing		√
20. Functional testing		√
21. Software metrics		√
22. Maintenance		
23. Reverse engineering		
24. Quality of software development process and its standardization		
25. Introduction to software ergonomics	√	
26. User manuals	√	
27. Project management		
28. Configuration and version management		
Assignment 1 – review of requirements specification document	√	
Assignment 2 – cost estimation	√	
Assignment 3 – review of the product model according to structured analysis	√	
Assignment 4 – derive a use case and class diagram for a new software specification		
Assignment 5 – derive a formal specification for a new software subsystem		
Assignment 6 – apply regression testing tool to a new small example program		√
Assignment 7 – build a classification tree for one use case	√	
Assignment 8 – apply some software metric tools to a new software		√

References

[1] ***, Joint Course on Software Engineering web site, http://www.informatik.hu-berlin.de/swt/intkoop/se/index.htm
[2] Balzert, Lehrbuch der Software-Technik, Spektrum Akademischer Verlag, - Vol. 1, 2nd Edition 2001 (in German).
[3] Preliminary requirement specification ver 3.0, http://www.informatik.hu-berlin.de/swt /intkoop/se/prelreqspec3_0.htm
[4] Requirement specification ver 3.0, http://www.informatik.hu-berlin.de/swt /intkoop/se/reqspec3_0.htm
[5] Preliminary requirement specification ver 2.3, http://www.informatik.hu-berlin.de/swt /intkoop/se/Sem_Org_Prel_Req_Spec.v23.htm
[6] Requirement specification ver 2.3, http://www.informatik.hu-berlin.de/swt /intkoop/se/Sem_Org_Req_Spec.v23.htm
[7] XCTL behavioral specification, http://www.informatik.hu-berlin.de/swt/intkoop/se/XCTL_Man_Adj.htm