

Topic 12

Documents

- The **prerequisite** for following this (part of the) lecture is that you have a basic awareness of the complexities of software engineering — such as, for example, outlined in earlier lectures.
- The **aims** are
 - ★ to introduce the concept of *documentation*,
 - ★ and to introduce the concepts of *informative*, *descriptive* (*prescriptive*, *specificational*) and *analytic* document parts and their many subparts.

- The **objective** is
 - ★ to make you a professional software engineer with respect to the crucial aspect of documentation.
- The **treatment** is informal, almost casual, but rigorous.

Documents and Projects

- Documents, such as they are produced during software engineering, belong to projects.
- Projects either develop domain descriptions, or requirements prescriptions or software specifications, or the first two, or the last two, or all three of these.
- Software engineering projects basically only develop documents and, less tangibly, human insight.
- In this lectures we shall be concerned with which kinds of documents are produced by software engineering projects.

Documentation Is All!

- The objective of software engineering is to construct (deploy and maintain) software.
- Earlier we gave a definition of what we mean by software.
- The essence of software engineering is to *construct* (and *analyse*) *documents* based on a context of people and the further environment in which this software is to be inserted.

- We mean not only the code (to be the basis for computations),
- but also all the other things listed in the characterisation of what software is syntactically.
- Software engineering does not construct material artifacts as is done in mechanical, civil, chemical, electrical and electronics engineering.
- It is therefore of utmost importance that the software engineer is competent wrt. all the aspects of documentation that we shall cover in this lecture.
- That is, it is important that the software engineer is a literate person.

Kinds of Document Parts

General

- We make the distinction between the following three kinds of document texts:
 - ★ texts which *inform*, but that do not (essentially) describe that which the phase of development is to develop,
 - ★ texts that *describe* (*prescribe*, *specify*) and
 - ★ texts that *analyse*.
- A development document may contain all of these document parts,
- but these parts should be
 - ★ clearly delineated
 - ★ and marked.
- parts of this lecture will now explain what we mean by these parts of proper documentation.

What Is a Description?

Information Versus Description Versus Analysis

- Above we made a distinction between three kinds of texts:
 - ★ texts that *inform*, but that do not (essentially) describe,
 - ★ texts that *describe* (*prescribe*, *specify*) that which the phase of development is to develop and
 - ★ texts that *analyse*.
- These distinctions need be clarified.

- But we warn the student: the distinctions can never be made fully transparent, fully unambiguous.
 - ★ There will be texts which can be claimed to be both descriptive and analytic,
 - ★ there will be texts which can be claimed to be both informative and descriptive (prescriptive),
 - ★ and so on.

- We later lecture
- outline the distinction already made, but not properly explained,
- between description and prescription.
- For the time being, please take them as semantically synonymous, but pragmatically distinct.

Characterisation 2.49 By a *describing (or prescribing) text* we mean

- a text that designates and/or delineates some physically existing phenomenon,
- or a text that defines a concept which can be said to be an abstraction of a physically existing phenomenon

.



Example 2.19 *A Rough Sketch Inner-City Street System*

Description:

- Let us imagine an inner-city collection of one-way streets.
- One may, for example, have written the following rough-sketch description of such a system:
 - ★ There is an area of a city called the inner-city.
 - ★ And there is a disjoint, but bordering area of the city called the outer-city.
 - ★ The city consists, trafficwise, of a finite number of streets.
 - ★ A street is something that has two ends, a length (between these), and which allows vehicles to move from one end to the other.
 - ★ Each inner-city street is one-way, i.e., allows traffic in only one direction.
 - ★ Each outer-city street is two-way, i.e., allows traffic in both directions.

- ★ Either an inner-city street is connected (has an intersection)
 - ◇ at each end to one or more inner-city streets,
 - ◇ or at each end to one or more outer-city streets,
 - ◇ or — in a blend of the above —
 - at one end to one or more inner-city streets, and
 - at the other end, to one or more outer-city streets.
- ★ A connection (an intersection) of two (or more) one-way streets (“wholly”) within the inner-city
 - ◇ is further restricted in such a way that any two such connected streets
 - ◇ allow traffic only in the same consecutive direction. .



Characterisation 2.50

By an *analysing text* we mean

- a text which proves (i.e., reasons over)
- or designates properties of some other text,
- or properties that are claimed to hold between pairs of texts

.



Example 2.20 *A Rough-Sketch Inner-City Street Analysis:* We continue Example 2.19 by stating properties that resulted from an analysis of the text of that example.

- It is not possible for traffic in the inner-city to go in circles.
- Once inside the inner-city it is indeed possible to also get outside.
- It is not possible to get both into and out from the inner-city at the same (two-street) connection.
- The maximum number of inner-city one-way streets that one may have to travel to get into and out from the inner-city is the number of inner-city one-way streets.



Characterisation 2.51 By an *informing text* we mean

- a text which is neither basically descriptive nor analytic,
- i.e., a text which does not designate physical phenomena or concepts directly related to these,
- but a text which otherwise “points” to or implies descriptive or analytic documents

- Whereas it is relatively easy to characterise what is meant by
 - ★ descriptive (prescriptive) and
 - ★ analytictexts,
- it is more difficult to characterise what is meant by
 - ★ informative texts.

We continue Examples 2.19–2.20. We give examples of informative texts that usually precede the texts of the rough sketches and their analyses.

Example 2.21 *A Rough-Sketch Inner-City Information:*

- The *partners* to this contractual work are (1) the government of (city) *X*, the department of roads, and (2) the informatics consultancy firm *Y*.
- The *current situation* is that the inner-city traffic is a mess.
- There is thus a *need* for an inner-city traffic system that can improve that situation.
- The *idea* is to provide for some kind of point-to-point traffic flow.

- That is, point-to-point traffic flow is a crucial *concept*. Street segments and street junctions are other crucial concepts.
- The *scope* of our concern is that of inner cities.
- The *span* of our concern is that of the traffic in their streets.
- We intend to *contract* a consulting firm strong in demographics, traffic analysis and prognosis to carry out a further study and come up with recommendations — including giving the city government a *design brief*.



- To make it easier for us to delineate what kind of informative texts are necessary in software development projects, we (thus) limit these texts as indicated above:
 - ★ *current situation, need, idea, concept, scope, span* and *contract*, including *design brief* texts.
 - ★ In addition we may also wish, at times, to formulate a *synopsis*, a kind of *summary* (a *resumé*) of the previous informative text parts.

- Thus the lecture part on informative documents will cover primarily these information topics.
- By providing this kind of “information document systematics” we, at the same time, make it easier for the software engineer to know that such informative documents are important, and what they should contain.

Descriptions, Prescriptions and Specifications

- We shall, consistent with later material on descriptions versus prescriptions, briefly make the following distinctions.
- They were already made, but were not fully explained:
 - ★ Domains are *described*,
 - ★ requirements are *prescribed*, and
 - ★ software is *specified*.
- It is useful to maintain these distinctions.

Deliverables

Characterisation 2.52 By a *deliverable* we shall understand a document which is called for in a contract and which is thus (to be) developed and is thus (to be) exchanged between the partners to that contract — as here software development: whether for just a domain description, or for just a requirements prescription, or for just a software design, or for parts thereof, or for combinations thereof. ■

- There are very many kinds of documents.
- Hence it is important that a contract makes it very clear which are the deliverables:
- their form, their expected contents, their expected degrees of detail and formality, whether they have been subject to validation, verification, tests, etc.
- As we shall also see, many documents may not (initially) be thought of as deliverables, yet their development serves crucial needs.

Topic 13

Informative Document Parts

- Before serious, usually long and, to some, tedious descriptions can be developed, let alone presented,
- the developers and their clients, must agree on some basics.
- Some common information must first be established.
- The informative documents serve this pragmatic purpose.

- We suggest the following kinds of informative document parts:

1. *name, place and date*

9. *synopsis*

2. *listing of partners*

10. *assumptions and dependencies*

3. *current situation*

11. *implicit/derivative goals*

4. *needs*

12. *standards*

5. *ideas*

13. *contract*

6. *concepts*

14. *design brief*

7. *scope*

15. *logbook*

8. *span*

- They may serve well to initially establish partner agreements.

Name, Place and Date

- A project must have a brief, informative name.
- A project takes place somewhere, in one or more locations.
- And an informative project document must be dated.

Partners

- Typically there can be many partners to contractual work in informatics.
 - ★ Those whom we typically would call the *clients*:
 - ◇ the enterprise, or institution, or others, who wish a problem to be investigated for possible solution by computing, and possibly
 - ◇ other such consultancy firms, or others, who may assist, incl., advise, the enterprise in the work with developers.

★ There are also those whom we accordingly would consider the *developers*:

- ◇ the prime developer, possibly also itself a consultancy firm, or, more typically, perhaps, a software house, and possibly
 - ◇ specialist consultancy firms, or research institution scientists who may assist, incl., advise, the prime developer in the work with the client.
- We will now treat these two categories a bit more.

Clients

- Clients can contract the development of
 - ★ domain models, inclusive of descriptive and analytic documents,
 - ★ or clients can contract the development of requirements models, inclusive of prescriptive and analytic documents, and as based on domain models.
 - ★ Clients can also contract the development of software designs, inclusive of specification and analytic documents, and as based on domain and requirements models.
 - ★ Clients can contract the development of combinations of domain and requirements models,
 - ★ or clients can contract the development of combinations of requirements models and software designs,
 - ★ or clients can contract the development of combinations of all: domain and requirements models, and software designs.

- To help the client in advising the developer and in evaluating the work (i.e., the deliverable documents) of the developer, the clients often use
 - ★ consultants,
 - ◇ either specialists in the domain of the client and in its informatisation,
 - ◇ or specialists in the quality assessment of the work of the developers.
 - ★ Lecturer explains an analogue from shipbuilding.

Developers

- Developers can, symmetrically, be contracted for the development of
 - ★ domain models, inclusive of descriptive and analytic documents,
 - ★ or requirements models, inclusive of prescriptive and analytic documents, and as based on domain models,
 - ★ or software designs, inclusive of specification and analytic documents, and as based on domain and requirements models.
 - ★ Developers can also be contracted for the development of combinations of domain and requirements models,
 - ★ or combinations of requirements models and software designs,
 - ★ or combinations of all: domain and requirements models, and software designs.

- To help the developer with the “harder” parts of the work, the developers may use consultants,
 - ★ either specialists in the domain of the client and in its informatisation,
 - ★ or specialists
 - ◇ in the verification of properties of the domain and requirements models, and of the software designs,
 - ◇ or in the verification of correctness of stages of development, from domains, via requirements to software,
 - ◇ or in specific, novel tool supports (i.e., development technologies).

Current Situation, Needs, Ideas and Concepts

Example 2.22 *Traffic Connection Across the Sea:*

- The example is taken from a field completely different from that of software. It concerns the possible establishment of non-ship connection between coastal points of two land masses.
 - ★ **Current situation:** The *current situation* is that a lot of people and vehicles: cars, trucks and trains must daily pass between these land masses, and that the time of waiting for and actual ferry transport imposes undue delays and these delays impose undue costs.
 - ★ **Needs:** There is therefore a *need* to speed up the waiting and transport times.

- ★ **Ideas:** The *idea* has therefore been put forward that a fixed connection be provided, either (a) a tunnel or (b) a bridge or (c) a combination of tunnel(s) and bridge(s).
- ★ **Concepts:** More specifically the following fixed connection *concept* has been settled upon, namely (i) the designation of two specific coastal areas, π_1, π_2 as the “anchor” points for the fixed connection, (ii) selected because there is a suitable small island, \emptyset between these two points, (iii) the provision of a low combined railway and road bridge between π_1 and some point on \emptyset , (iv) the provision of a high road bridge between another point on \emptyset and π_2 , and, finally, the provision of a railway tunnel between another point on \emptyset and π_2 .

- Notice how the issues of the current situation feed into the needs, how the needs feed into the idea, and how the idea leads to albeit loosely formulated concepts.
-
- Example 2.22 expresses a meta-requirement, namely (the idea) that a fixed connection replace ferries.
- Transferring the above example to software engineering and hence to any of the three phases, the meta-requirement either expresses the ideas of either better understanding the domain, or obtaining a requirements prescription, or designing software.
- In Example 2.22 there was no expression of any specific requirements to the fixed connection such as being able to carry so-and-so-much road and rail traffic at such-and-such costs etcetera.

Current Situation

Characterisation 2.53 By a *current situation* — in the context of informative software development documentation — we shall understand

- an informal expression of
- problems of an area of the universe of discourse:
 - ★ be it in connection with a domain,
 - ★ or in connection with requirements,
 - ★ or in connection with software design

.



Example 2.23 *Current Railway Net Handling Problem:*

- The handling of railway net maintenance is very costly.
- Handling procedures are apt to take a long time and yet be erroneous.
- Handling is, amongst others, based on the human manipulation of a huge repository of paper documents and drawings concerning the railway net and its actual status (whether worn, past history of repairs, etc.).



- The current situation that may motivate just a domain description project — one that may not necessarily be followed by a requirements prescription project — is usually that it is hard to bring new, untrained staff into a domain. There is simply no good introductory training material.
- The current situation that may motivate a requirements prescription project is usually that software has to be developed.

Needs

Characterisation 2.54 By a *need* —

- in the context of informative domain description documentation — we shall understand
 - ★ an informal expression of the need for a better understanding of the domain — and —
- in the context of informative software requirements prescription documentation — we shall understand a need as
 - ★ an informal expression of the need for software, and hence for a requirements prescription

.



Example 2.24 *Need for Improvement of Railway Net Handling:*

We continue Example 2.23.

- There is, following the identification stated in Example 2.23, thus
- a need to significantly improve railway net maintenance handling costs, time and accuracy.



Ideas

Characterisation 2.55 By an *idea* — in the context of informative software development documentation — we shall understand

- a very brief formulation of how, in principle, the *need* — in the context of software development —
- may be fulfilled

.



Example 2.25 *Need for Railway Net Support Software:* We continue Example 2.24.

- The idea is to computerise railway net documentation and its handling procedures,
- to do so by means of a distributed computing system
- with transparency between “where stored” and “where used”,
- and to have the system readily store, display and search over texts, drawings, measurements (tables, curves) and photos.



- Current problems and hence derived needs may be fulfilled by means other than computing systems:
 - ★ New management and/or better trained staff, or
 - ★ new handling procedures (i.e., business process reengineering),
 - ★ or other.

- The idea, usually, in the context of a domain description project, is to develop exactly that: a domain description.
- And the idea, then, in the context of a requirements prescription project, is to develop exactly that: a requirements prescription.
- But these ideas may be expressed in a more elaborate, informative form than saying “just that”!

Concepts and Facilities

- The pragmatics of the “Concepts and Facilities” section
- is to — ever so briefly — inform all parties to the contract of
- which are the most important ideas of the subject domain of the contract.

Characterisation 2.56 • We use the terms *concepts* and *facilities* more or less interchangeably:

- ★ A facility is a physical phenomenon
- ★ while a concept is a mental construction (covering, usually some physical phenomena or concepts of these).

- The concepts and facilities referred to here, in the informative parts of project documentation, depend on the universe of discourse.
 - ★ In the context of informing only about a domain description development project
 - ◇ the concepts and facilities are intended,
 - ◇ in the document section of that name,
 - ◇ to be the most pertinent of the domain.
 - ★ In the context of informing only about a requirements prescription development project
 - ◇ the concepts and facilities are intended,
 - ◇ in the document section of that name,
 - ◇ to be the most pertinent of the requirements

.

■

- Several concepts and facilities may have to be mentioned in the “early” information documents of a project.

Example 2.26 *Concepts and Facilities of Railway Net*

Computerisation: We continue Example 2.25.

- The concepts and facilities include:
 - ★ To further represent each railway net unit, i.e., each smallest individually handled part of a net, as a tuple, i.e., as a separately storable entity in a relational (say SQL) database;
 - ★ to represent relations between (neighbouring, etc.) such units, and units and adjacent (separately, also relation represented) signals, power line wiring, platforms, etc., by similar SQL relations, etc.;

- ★ to also represent civil, mechanical, electromechanical, and electronics engineering images. By these we mean drawings, measurements and photos such that these images can be referred to by suitable attributes — possibly annotating these images in the form of SQL relations.
- ★ The conceptual ideas are to provide for a geographical distribution of such databases,
- ★ to provide relations of all of the above to a geographical information system (a GIS) for the landscapes of the rail nets,
- ★ and to provide means of exchange between railway maintenance centres of such data as outlined above by means of XML.



- The concepts, in the context of a domain description project, amount to a listing of major entities, functions, events and behaviours of the domain to be described.
- Possibly the domain description concepts also include a listing of the major intrinsic, support technology, management and organisation, rules and regulation, etcetera, phenomena and concepts.

Scope, Span and Synopsis

- The four kinds of informative document parts:
 - ★ current situation,
 - ★ needs,
 - ★ ideas, and
 - ★ concepts,
- form an introductory “whole”
- that now need be “solidified”.
- They need to be brought together in a more coherent fashion — in what we shall call the
 - ★ scope, span and
 - ★ synopsis document

Scope

Characterisation 2.57 By *scope* — in the context of informative software development documentation — we shall understand

- an outline of the broader setting of the problem, i.e., the universe of discourse at hand

.



Example 2.27 *Scope of Transportation:* We rough-sketch characterise an infrastructure component.

- The problem, in general, is to understand the domain of transportation:
- of road, rail, air and sea transport, and of
- logistics of multi-modal freight transport.



Span

Characterisation 2.58 By a *span* — in the context of informative software development documentation — we shall understand

- an outline of the more specific area
- and the nature
- of the problem that need be tackled

.



Example 2.28 *Span of Railway Systems:*

- The problem whose scope was outlined in Example 2.27 is, more specifically, to build a theory of the domain of railway systems:
- of rail nets, trains, traffic, its planning, monitoring and control, of passengers and freighters, etc.



Synopsis

Characterisation 2.59 By a *synopsis* — in the context of informative software development documentation — we shall understand the same as a resumé, a summary, that is,

- a comprehensive view, that is,
- an extract of a combination of current situation, needs, ideas, concepts, scope and span documentation
- informing about a universe of discourse for which some development work is desired:
 - ★ the construction of a domain description,

- ★ or the construction of a requirements prescription based on an existing domain description, or both;
- ★ or the construction of a software design based on existing requirements prescription;
- ★ or both (requirements and software design),
- ★ or all (domain, requirements and software design);
- ★ or the first two (domain and requirements)

Example 2.29 *Synopsis for a Railway Systems Domain Theory Project:*

- The project is to develop and research a domain model for a railway system — such as scope and span indicated above!
- Thus the domain model is expected to cover such phenomena as:
 - ★ the railway net: its static and dynamic properties, including signaling;
 - ★ train timetables: their planning, construction and as basis for traffic;
 - ★ the rolling stock: its composition onto, and decomposition from trains, maintenance, etc.;
 - ★ traffic: the dispatch, monitoring and control of trains, including rescheduling, etc.;
 - ★ the railway staff: Their rostering onto station, net and train service;
 - ★ and so on.



- There is, of course, much more to a synopsis than exemplified above.
- There is more text on scope and span, and more details than just exemplified.
- Typically between half a page and two thirds of a page. Usually no more. Signatories to a contract will not read any larger such texts!



- Two hard-to-quantify kinds of informative text are often in the backs of the heads of those who procure computing systems.
 - ★ One kind is the assumptions that are made about that which lies outside the domain of the application, and the dependencies any such required computing system will later experience.
 - ★ The other kind are the meta-requirements that are expected to be fulfilled by the deployment of the required computing system.
- They are treated next.

Assumptions and Dependencies

- There are two kinds of assumptions and dependencies.
 - ★ One kind has to do with sources of knowledge.
 - ◇ For domain development there needs to be the sources from which the domain engineer can learn about and develop the domain description.
 - ◇ For requirements development there needs to be a domain description as well as people from whom the requirements engineer can elicit the requirements and thus develop the requirements prescription.
 - ◇ And for software design there needs to be a requirements prescription.

★ The other kind has to do with delineation of the domain.

- Usually a domain description
- (one upon which we base our (domain) requirements)
- leaves out
- what we might call the “fringes” of the domain,
- i.e., the environment of that domain.
- To also describe those parts might simply “be too much”!
- That environment is simply judged too large, too unwieldy, to describe.

- Yet, sooner or later, that environment will show up in the requirements prescription, if it is not already in the domain description.
- The requirements prescription eventually, thus, comes to depend — maybe not exactly crucially, but anyway — on events originating in the environment, or the ability of the computing system to dispose of some output to that environment.
- Loosely, we admit, that is what we mean by assumptions and dependencies.
- We shall return to this issue when we review requirements prescriptions with respect to their being faithful to assumptions and dependencies.

Example 2.30 *Two Sets of Examples of Assumptions and Dependencies:*

- For the development of a domain description of the financial services industry
 - ★ it is *assumed* that the developers have access to a necessary and sufficient number of professionals possessing the necessary and sufficient knowledge about respective parts of their domain, and
 - ★ the quality of the resulting domain model (also) *depends* on fulfillment of the assumptions.

- For the development of a requirements prescription for, say a specific bank within the financial service industry domain
 - ★ three sets of *assumptions* are expected to be fulfilled:
 - ◇ that there is a mutually accepted domain description of the relevant parts of the financial services industry,
 - ◇ that the developers have access to a necessary and sufficient number of professionals possessing the necessary and sufficient knowledge about respective parts of desired requirements and
 - ◇ that the client is ready and willing to consider possible business process reengineerings of the current management and operations of the domain.

- ★ Two sets of *dependencies* are expected to be fulfilled:
- ◇ the quality of the resulting requirements model (also) *depends* on fulfillment of the assumptions,
 - ◇ and the users of the eventually developed and installed software loyally adopt possible business process reengineering practices.
-

Implicit/Derivative Goals

- Usually computing systems provide, or offer, a large number of entities, functionalities, events and behaviours,
- and it is those requirements we prescribe.
- But those entities, functionalities, events and behaviours really do not themselves reveal why they are or were prescribed.
- Usually their prescription serves “ulterior” goals which cannot be quantified in a way that indicates what the prescribed computing system should offer.

- Typical metagoals are such as:
 - ★ “Deployment of the computing system should result in greater profits for the company.”
 - ★ “Deployment of the computing system should result in the company attaining a larger market share for its products.”
 - ★ “Deployment of the computing system should result in fewer worker accidents.”
 - ★ “Deployment of the computing system should result in more satisfied customers (and staff).”

- Other kinds of metagoals are:
 - ★ “The existence of a domain description will have led or should lead to better understanding of the domain, hence to improved performance of domain staff trained in the domain based on such domain descriptions.”
 - ★ “The existence of a requirements prescription will have led or should lead to more appropriately targeted software.”

Example 2.31 *Two Sets of Examples of Implicit/Derivative Goals:*

- For the development of a domain description of the financial service industry we foresee the following implicit or derivative goals:
 - ★ better training of new banking staff (as based on a trustworthy domain description),
 - ★ a better foundation for discussion of business process reengineering plans for the financial service industry (say due to new government regulations), and
 - ★ improved confidence in tackling ambitious computerisation plans.

- For the development of a requirements prescription for, say a specific bank within the financial service industry domain we foresee the following implicit or derivative goals:
 - ★ (although explicit requirements cannot be directly implemented as part of the desired computing system)
 - ★ it is hoped that the requirements (to be developed) can lead to a computing system that helps the bank attain a leading-edge position in the industry,
 - ★ while securing more enjoyable staff working conditions
 - ★ and improved profits.



Standards

- A distinction is made between development standards and documentation standards.

Development Standards

- Usually development occurs in the context of following some development standards (one or more).
- The Institute of Electrical and Electronics Engineers (IEEE) has established a number of standards for the development of a various kinds of software.
- Other national and international organisations, including the International Organization for Standardization (ISO) and the International Telecommunication Union (ITU), have established similar standards.

Documentation Standards

- Usually documentation occurs in the context of following some documentation standards (one or more).
- The Institute of Electrical and Electronics Engineers (IEEE) has established a number of standards also for the documentation of a various kinds of software.
- Other national and international organisations, including the International Organization for Standardization (ISO) and the International Telecommunications Union (ITU), have also established similar standards.

Standards Versus Recommendations

- Some standards are binding, some are recommendations.
- Reference to specific standards and recommendations can be written into project contracts with the meaning that the project must comply with these standards and recommendations.
- Some standards mandate or recommend the use — and hence the documentation style — of certain development practices.
- Other standards mandate or recommend the use of specific spelling forms, mnemonics, abbreviations, etc.

Specific Standards

- There are very many standards for software development and for its documentation.
- Some standards come and go. Others are quite stable.
- A study of more specialised standards reveals the following acronyms: MIL-STD-498, DOD-STD-2167A, RTCA/DO-178B, JSP188 and DEF STAN 05-91.
- The course student is invited to search for these on the Internet.
- It therefore makes little sense for us to list other than a few clusters of seemingly more stable and trustworthy standards.

★ *International Organization for Standardization (ISO):*

<http://www.iso.ch/>

- ◇ ISO 9001: Quality Systems Model for quality assurance in design, development, production, installation and servicing
- ◇ ISO 9000-3: Guidelines for the application of ISO 9001 to the development, supply and maintenance of software
- ◇ ISO 12207: Software Life Cycle Processes
<http://www.12207.com/>

★ *IEEE Standards*: <http://standards.ieee.org/>

◇ IEEE Std 610.12-1990, Standard Glossary of Software Engineering Terminology

This standard contains definitions for more than 1000 terms, establishing the basic vocabulary of software engineering.

◇ IEEE Std 1233-1996, Guide for Developing System Requirements Specifications

This standard provides guidance for the development of a set of requirements that, when realized, will satisfy an expressed need.

◇ IEEE Std 1058.101987, Standard for Software Project Management Plans

This standard specifies the format and contents of software project management plans.

◇ IEEE Std 1074.1-1995, Guide for Developing Software Life Cycle Processes

This guide provides approaches to the implementation of IEEE Std 1074. (This standard defines the set of activities that constitute the mandatory processes for the development and maintenance of software.)

◇ IEEE Std 730.1-1995, Guide for Software Quality Assurance Plans

The purpose of this guide is to identify approaches to good Software Quality Assurance practices in support of IEEE Std 730. (The standard establishes a required format and a set of minimum contents for Software Quality Assurance Plans. The description of each of the required elements is sparse and thus provides a template for the development of further standards, each expanding on a specific section of this document.)

◇ IEEE Std 1008-1987 (reaffirmed 1993), Standard for Software Unit Testing

The standard describes a testing process composed of a hierarchy of phases, activities, and tasks. Further, it defines a minimum set of tasks for each activity.

◇ IEEE Std 1063-1987 (reaffirmed 1993), Standard for Software User Documentation

This standard provides minimum requirements for the structure and information content of user documentation.

◇ IEEE Std 1219-1992, Standard for Software Maintenance

This standard defines the process for performing the maintenance of software.

- ★ *Software Engineering Institute (SEI):* <http://www.sei.cmu.edu>
 - ◇ *Software Process Improvement Models and Standards, including SEI's various Capability Maturity Models*
- ★ *UK Ministry of Defence Standards* <http://www.dstan.mod.uk/>
 - ◇ 00-55: Requirements for Safety Related Software in Defence Equipment
<http://www.dstan.mod.uk/data/00/055/02000200.pdf>
 - ◇ 00-56: Safety Management Requirements for Defence Systems
<http://www.dstan.mod.uk/data/00/056/01000300.pdf>
- So, please, use the Internet for latest on standards relevant to your project.

Contracts and Design Briefs

Contracts

- The
 - ★ current situation, needs, ideas, concepts,
 - ★ scope, span and synopsisdocument parts are
 - ★ preambles to, set the stage for, and are a necessary background forcontractual documents.
- Usually one contract (document) is sufficient for small projects.
- And usually several related contracts (documents) are needed for larger projects.

Characterisation 2.60 By a *contract* — in the context of informative software development documentation — we shall understand a separate, clearly identifiable document

- which is legally binding in a court of law,
- which identifies parties to the contract,
- which describes what is being contracted for possibly mutual deliveries, by dates, by contents, by quality, etc.,
- which details the specific development principles, techniques, tools and standards to be used and followed,
- which defines price and payment conditions for the deliverables,
- and which outlines what is going to happen if delivery of any one deliverable is not made on time, or does not have the desired contents, or does not have the desired quality, etc

- For national and for international contracts predefined forms which make more precise what the contracts must contain are usually available.
- We will not bring in an example.
- Such an example would have to reflect the almost ‘formal’ status of ‘legal binding’, and would thus have to be extensive and very carefully worded, hence rather long.
- Instead we refer to national and international contract forms.
- The software development field is undergoing dramatic improvements.
- Clients are entitled to have legally guaranteed quality standards (incl. correctness verification).

- Hence contracts will have to refer to
 - ★ the broader domain and give specific references to named domain stakeholders, if the development of a domain description is (to be) contracted; or
 - ★ existing domain descriptions and give specific references to named stakeholders, if the development of a requirements prescription is (to be) contracted; or
 - ★ existing requirements prescriptions and give specific references to named stakeholders, if the development of software is (to be) contracted.
- Therefore contracts should name “the methods” by means of which the deliveries will be developed — as we have indicated in bullet four of the characterisation.

Design Briefs

Characterisation 2.61 By a *design brief* we understand

- a clearly delineated subset text of the contract.
 - ★ This text (bullet three) describes what is being contracted for possibly mutual deliveries, by dates, by contents, by quality, etc., and
 - ★ (bullet four) it details the specific development principles, techniques and tools;
- that is, the design brief directs the developers, the providers of what the contract primarily designates,
- as to what, how and when to develop what is being contracted

Example 2.32 *A Design Brief:*

- The supplier is to develop
 - ★ a domain description of
 - ★ the equipment procurement part
 - ★ of company X ,
- by such-and-such a date, using best practices domain description principles, techniques and tools — such as illustrated in such-and-such a textbook,
- etc., etc.



Logbook

Characterisation 2.62 *Logbook*: By a *logbook* we understand

- a record, a set of notes,
- which as correctly as is humanly feasible,
- lists the
 - ★ development, release, installation, use, maintenance, etc., history

A logbook serves as a necessary reference in innumerable, usually unforeseeable instances of development.

Example 2.33 Logbook: A postulated logbook may reveal:

2 Jan. 1991: Initial meeting between partners $\mathcal{E}c$.

...

31 May 1993: Acceptance of domain model $\mathcal{E}c$.

...

24 October 1994: Acceptance of requirements model $\mathcal{E}c$.

...

3 June 1996: Acceptance of software delivery $\mathcal{E}c$.

...

The $\mathcal{E}c$. signify reports, and the ... signify other logbook entries. ■

Discussion of Informative Documentation

General

- We have identified some useful components of informative document parts.
- There may be other such informative parts. It all may depend on the universe of discourse, i.e., the problem at hand.
- We thus encourage the software developer to carefully reflect on which are the necessary and sufficient informative document parts.
- There is usually a separate set of informative documents to be worked out for each phase of development:
 - ★ the domain phase,
 - ★ the requirements phase, and
 - ★ the software design phase.

- The current situation, needs, ideas, concepts, scope, span, synopsis and contract document parts differ in content between these phases.
- Usually the informative document parts, although crucially important, need not require excessive resources to develop, but their development must still be very careful!
- In general, the informative document parts are concerned with the socioeconomic, even geopolitical, and hence pragmatic context of the projects about which they inform.
- As such they are “fluid”, i.e., less precise, in what they aim at and what their objectives are.
- The next two documentation kinds are, in that respect, much more precise, and much more focused.

Methodological Consequences: Principle, Techniques and Tools

Principle 2.1 *Information Document Construction:*

- When first contemplating a new software development project,
- make sure — as the very first thing —
- to establish a proper complement of (all) informative documents.
- Throughout the entire development and after —
- during the entire lifetime of the result,
 - ★ whether a domain model,
 - ★ or a requirements model,
 - ★ or a software system —
- maintain this set of informative documents

Principle 2.2 *Information Documents:*

- The informative documents must be authoritative,
- definitive and
- interesting to read

.



Techniques 1 *Information Document Construction:*

- First establish a document embodying the fullest possible table of contents, whether for
 - ★ just a
 - ◇ domain development, or a
 - ◇ requirements development, or a
 - ◇ software design project,
 - ★ or for a combination of these.

- Then fill in respective document parts,
 - ★ “little by little”, just a few sentences,
 - ★ using terse, precise, i.e., concise language,
 - ★ while avoiding descriptions (prescriptions and specifications) and analyses.
 - Throughout maintain clear monitoring and control of all versions of these documents
-

Tools 2.1 *Information Document Construction:*

- A text processing system, preferably L^AT_EX, but MS Word will do,
- with good cross-referencing facilities, even between separately ‘compilable’ documents,
- provides a ‘minimum’ tool of documentation.
- Add to this a reasonably capable version monitoring and control system (such as CVS) and you have a workable system

Topic 14

Descriptive Document Parts

- To work out, i.e., to develop the descriptive documentation, for any phase, is, in contrast to work on informative documentation, a major and therefore a resource-critical development effort.
- We remind the student of our earlier characterisation of the concept of description:

Characterisation 2.63 By a *describing (or prescribing) text* we mean

- a text which designates and/or delineates some physically existing phenomenon,
- or a text which defines a concept which can be said to be an abstraction of a physically existing phenomenon

.



- We shall consider four kinds of descriptions:
 - ★ rough sketches,
 - ★ terminologies,
 - ★ narratives and
 - ★ formalisations.

Informal and Formal Document Parts

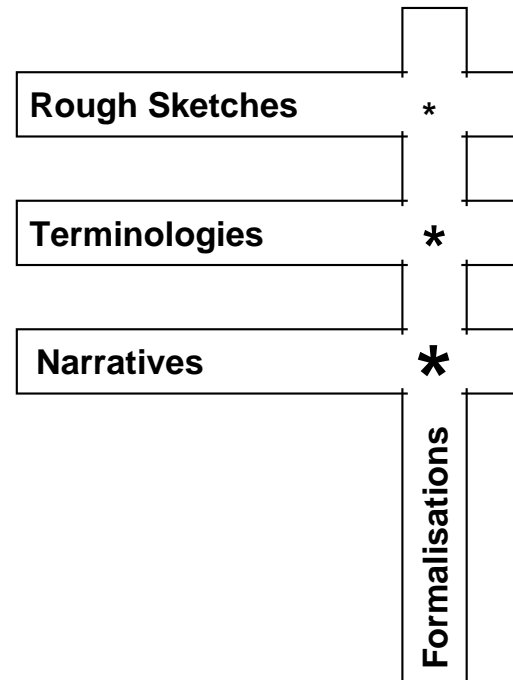


Figure 2.5: Informal and formal document texts

Rough Sketches

Characterisation 2.64 *Rough Sketch*: By a *rough sketch* — in the context of descriptive software development documentation — we shall understand a document text

- which describes something
- which is not yet consistent and complete,
- which may still be too concrete, overlapping, or repetitive in its descriptions, and
- with which the describer has yet to be fully satisfied

.



Pragmatics, Semantics and Syntax of Rough Sketches

- The *pragmatics* of rough sketches is that their construction serve as the first manifest step of a development, and that the rough sketches can be subject to analysis, and hence the formation of concepts.
- The *semantics* of rough sketches is hazy. Rough sketches are indeed intended to denote something, but being rough, and being sketches may mean that there are many loose ends (i.e., incompleteness), some inconsistencies.
- The recommended *syntax* of rough sketches, in their informal, typically, as here, English form, is intended to follow the following lines: Separate, in clearly marked paragraphs, the sketching of entities (types and values), functions, events and behaviours. Their order is not prescribed. More will be said later.

Discussion

- Rough sketching, as a bona fide activity
- (although rough sketches are usually not part of deliverables),
- allows the developer to get started.
- Such starts represent a less systematic way than subsequent analysis and further descriptions may afford.

Example 2.34 *Rough Sketch of Freight Logistics:* By freight logistics we mean a structure of entities, of functions, events and behaviours that include:

- senders of freight: inquiring about freight despatch conditions, submitting freight for conveyance, tracing such freight, etc.;
- receivers of freight: inquiring about freight arrivals, fetching such freight, etc.;
- logistics firms which respond to inquiries (as above), which receive and deliver freight from senders, respectively to receivers; which arrange with transport companies for the transport of freight; which trace (and organise the rerouting of) freight; etc.;
- transport companies, such as trucking firms, train operators, shipping lines and airlines, that is, companies which operate line or bulk transfer of freight, whose trucks, trains, ships, and aircraft call at truck and train depots, harbours and airports for transfer of freight, etc.;
- truck and train depots, harbours, and airports that provide facilities for temporary storage and the receipt and the handing out of freight, etc.



- The above is a description in that certain phenomena of a physical world — entities, functions and behaviours — are identified, and certain relations between them are hinted at.
- It is a rough-sketch description in that the description raises more questions to be answered than before the student had read the description.
- The rough sketch description, as an activity, is therefore to be followed by the *analytic activity of concept formation*.

- Rough sketching is an iterative, explorative and experimental activity.
- The developer tries one approach, and analyses the result, and may not be happy.
- So the developer tries another rough-sketch approach, analyses the result, and may still not be happy.
- Hopefully this exploration of alternative approaches, experimentation with different starting points, converges.
- It is crucial that the developer is willing to scrap, or throw away, rough sketches that are not felt conducive to concept formation analysis, that is, for which such analysis does not yield interesting, illuminating concepts.

- Rough sketching is an indispensable activity the more research-oriented a development project is.
 - ★ Some development projects, whether of domain descriptions, requirements prescriptions, or a software design, are new:
 - ★ the developers (at least) have no or little experience in the field.
 - ★ Hence experimental or explorative development is required.
 - ★ For such research oriented development one cannot expect fixed, a priori determinable allocation of resources, including staff and time!
- The need for research-oriented development is often overlooked.

Methodological Consequences: Principles, Techniques and Tools

Principle 2.3 *Rough Sketching*: When embarking on a new development, of a domain description, of a requirements prescription, or a software design,

- first do some rough sketching,
- present this to colleagues,
- get comments,
- revise the rough sketches,
- etc., until ready for terminologisation and concept analysis

Techniques 2 • *Rough sketching* — usually based, as we shall see much later in these lectures, on domain or requirements acquisition — is, despite the acquisition, still an art.

- It requires creative skills.
- It can probably best be learned, by reading many rough sketches (and, subsequently, many narratives).
- Trial and error with peer review is a good way of learning rough sketching.

- Loosen up, relax, go for a walk, think of the topic.
- Try to sketch, and if you are not satisfied, throw it away — do not try editing an unsatisfactory sketch. Start all over. Remember that the essence is inside your head.
- If you face continued problems, during a day's work, leave it for that day. Think of it on your evening stroll, before going to bed; think of it while you try fall asleep. In the early morning it will all be much more clear to you!
- Rough-sketching is not a group effort; it is not a committee task.
- Individuals have to show the way

.



Terminologies

Characterisation 2.65 By a *terminology* — in the context of descriptive software development documentation — we shall understand a document text

- which is a collection of term entries,
- where a term entry is a pair: the term to be defined and its (narrative) definition,
- such that these definitions may contain other terms that need be defined,
- but such that a terminology claimed complete has all terms defined and none with unresolved circularities

.

Principle 2.4 *Terminologisation*:

- In any development project phase
 - ★ establish, early in the phase, a terminology for the universe of discourse,
 - ★ adhere, in all documentation, to this terminology and
 - ★ maintain the terminology.
 - ★ That is, make sure that
 - ◇ needs for updates, i.e., changes, are immediately met,
 - ◇ and that such changes are immediately reflected in all documentation

Pragmatics, Semantics and Syntax of Terminologies

- The *pragmatics* of a terminology is given just above, under terminologisation.
- The *semantics* of a terminology is that every term is well-defined and defines something!

- The recommended *syntax* of a terminology is suggested to be such as to help facilitate the following operations that terminology creators, users, and maintainers typically perform on terminologies:
 - ★ Separating, i.e., setting out, terms that are terms of a terminology from words that are commonly used, but are not terms of the terminology, that is: Finding out whether a word is a term.
 - ★ Finding out whether a term has been defined.
 - ★ Finding all uses of a term in other definitions.
 - ★ Identifying terms that are used but not defined.
 - ★ Somehow listing, i.e., displaying, the structure of all term definitions invoked by a given term definition.

Terminologisation — Continued

- The terminologisation principle basically requires that separate terminologies are (i) established, (ii) adhered to and (iii) maintained, for each phase:
 - ★ for the professional terms of the application domain,
 - ★ for the professional terms of the requirements domain and
 - ★ for the professional terms of the software technology domain.

Example 2.35 *Railway System Terminology*: We bring in but a fragment of a terminology:

- A **rail connector** is a further unexplained property of a *rail unit*.
- A **rail line** consists of one or more **linear rail units**,
 - ★ such that these are *sequentially* connected,
 - ★ and such that the *first* and *last rail units* of any **line** are *connected* to **rail units** of a **station**.

- A **rail unit** is a smallest entity of railway signalling concern.
- As properties a rail unit may have either of the following:
 - ★ A rail unit is either *linear* (straight or curved), or is a *switch*, or is a *crossover*, etc.
 - ★ A rail unit has *rail connectors* by means of which it may be *connected* to other rail units.



- In the above example we have hinted at
 - ★ a need for a proper structuring and typography of each entry:
 - ★ Some terms are *designatable*, others are *definable*.
 - ★ Many other remarks could be made about terminologies.
- We shall later explain the distinction between designations and definitions.
- We have listed the (only) three entries alphabetically.

Methodological Consequences: Principles, Techniques and Tools

- Constructing a terminology is an art.
- Some basic principles must be adhered to.
- Techniques and tools can be suggested.
- We previously enunciated a principle of terminologisation.
- We will, next, repeat a version of this principle,
- and then follow up with corresponding techniques and tools.

Principle 2.5 *Terminologisation*:

- Main terminology construction principles are:
 - ★ Every term of the professional language of the terminologised universe of discourse must be included in the terminology.
 - ★ And every term must be defined only using ordinary, i.e., commonly agreed-upon natural language and terms of (i.e., defined by) the terminology

Techniques 3 Terminology Documentation:

- Some terminology construction techniques are:
 - ★ First identify a smallest, or a reasonably small set of terms whose definition need not involve defined terms.
 - ★ When defining terms formally identify the denotation, i.e., the mathematical, not necessarily the computational, meaning entity τ , of the atomic term.
 - ★ Then define remaining, i.e., composite, terms using one or more of (already, or to be) defined terms $(t_1, \tau_1), (t_2, \tau_2), \dots, (t_n, \tau_n)$.
 - ★ When defining composite terms formally, identify whether there naturally exists a homomorphic function H by means of which the meaning of the compositely defined term, m , can be given as $H(\tau_1, \tau_2, \dots, \tau_n)$.
 - ◇ If so, then define the term homomorphically.
 - ◇ If H is not easily found, then define the term operationally



Tools 2.2 Terminology Documentation:

- The tools mentioned in Sect. apply equally well here.
- The ontology school referred to above is represented by a number of tools.
 - ★ These are, however, all experimental.
 - ★ So we will just mention them in passing and otherwise alert the reader to follow up, via the Internet or otherwise, searching for the “latest” ontology creation tools

Narratives

Characterisation 2.66 By a *narrative* — in the context of descriptive software development documentation — we shall understand a document text

- which, in precise, unambiguous language,
- introduces and describes all relevant properties of
 - ★ entities, functions, events and behaviours
- of a set of phenomena and concepts,
- in such a way that two or more readers
- will basically obtain the same idea as to what is being described

.

Pragmatics, Semantics and Syntax of Narratives

- The main purposes of constructing, i.e., the *pragmatics*, of narratives are:
 - ★ (i) To secure that those who develop the narrative indeed do understand what they document: the domain, the requirements, or the software design.
 - ★ (ii) To communicate to stakeholders that which is documented: the (described) domain, the (prescribed) requirements or the (specified) software design.

- ★ These stakeholders include, for domain descriptions and requirements prescriptions: various groups within the contracting client and, in general, within the domain.
- ★ For software designs the stakeholders include other groups of software developers than those who wrote the software design specification.
- ★ Of course, a mere narrative description is no guarantee of understanding, but its acceptance by other stakeholders brings an assurance of understanding rather much closer.

- The *semantics* of narratives is that they designate something, either physically or mathematically manifest, i.e., narratives describe something!
- The *syntax* of narratives typically is structured around clearly identified paragraphs that describe entities, functions, events and behaviours.
- A narrative, to be fully satisfactory,
- is paired with a mathematical model (say, specified in **RSL**).
- The narrative can be said to be an informal rendering of the formal model.

Example 2.36 *A Document Narrative:* We introduce a new kind of domain: that of **documents**.

- We rely on three basic, further unexplained notions: **text**, **location**, and **time**.
- By a document we understand some text.
- A document can be **created**, **edited**, **copied** or **deleted**. That is, a document is either an **original**, a **version**, or a **copy** or a former document no longer exists.

- From a document one can observe

- ★ its **text**,

- ★ and **references** to the

- ◇ **location**

- ◇ and **time**

at which the document was either **created** (as for an **original**), or **edited** (as for a **version**) or **copied** (as for a **copy**).

- There are, in all, seven operations which involve and/or result in documents. They are:
 - ★ (i) **Creating** a document, which constructs such a document “essentially” from “nothing”!
 - ★ (ii) **Editing** a document, which takes a document and results in a version of the “same” document such that it is always decidable which editing was done to the input, i.e., the underlying document.
 - ★ In other words, one can observe both the document (and hence text) of the version before and after editing.

- ★ (iii) **Tracing** the origins of a document: from a version and from a copy one can observe the document from which the version was edited, respectively copied.
- ★ (iv) **Copying** a document, which takes a document and results in two things, the unchanged document from which the copying was made, and the copy, which is a document with basically the same text.
- ★ (v) **Moving** a document, over some time, from some location to some other location, where it is assumed that the document “has rested” and “will rest” at the “from”, respectively the “to” locations.
- ★ (vi) **Shredding** (deleting) a document: effectively “removing it from existence”.

- ★ (vii) **Comparing** two documents: There are basically two kinds of comparisons, one is manifest, the other is conceptual, but cannot be effected!
- ◇ At any one time one can compare two documents to test whether they have the same origin, i.e., are “descended”, through copying and “versioning”, from a same (not necessarily original) document.
- ◇ And one can, figuratively speaking, compare two documents, where one document reflects a status at some time t and the other document reflects a status at some therefrom distinct time t' .
- ★ The idea behind the (unresolved) comparison operation is to be able to speak of two documents at distinct times “being the same” (or being related through suitable copying and versioning).

- Some axioms and derivable properties are needed:
 - ★ No two documents can occupy the same location at any one time, hence no two documents can be created at the same time and location.
 - ★ Any one document occupies exactly one location at any one time, and hence cannot be copied, or edited, at the same time in two or more different locations.
- We can think of a document system, i.e.,
 - ★ a collection of zero, one or more documents,
 - ★ as a function from time to functions from locations to documents.
- Such a document system has a **time of origin**, when the first document was first created.

- Further axioms and properties:
 - ★ A document which “exists” (as such) at two distinct times in a document system, exists at all times in between those two times, that is, there are no “ghost documents”.
 - ★ The sum total, at any one time, of documents of a document system is the sum total, since the time of origin to the present time, of
 - ◇ the number of document creations,
 - ◇ plus the number of “copyings”,
 - ◇ minus the number of shreddings.
 - ★ That is, documents do not suddenly disappear, never to reappear.
- $\mathcal{E}c$.



Discussion

- Constructing pleasing narratives is, perhaps, the most important part of development.
- Writing these narratives usually requires a major part of the development resources.
- The last step of development is that of “turning” narratives — or their formalisations — into executable code.
- In between, but not illustrated in this early lecture, narratives, from first being fully textual, increasingly contain diagrammatic as well as pseudo-program, or even formal texts.
- Later, much later lectures will illustrate the smooth transition from “plain texts” to texts increasingly containing diagrams or formalisations.

Methodological Consequences: Principles, Techniques and Tools

Principles 2.6 • The main principle of *narration* is to describe (prescribe, specify):

- ★ to only narrate things (phenomena, concepts)
 - ◇ that exist (as in the domain),
 - ◇ that shall exist (as for requirements), or
 - ◇ that will exist (as for software design).

★ A derivative principle of *narration* is to keep what is described (prescribed, specified) apart, including:

- ◇ domain facts,
- ◇ requirements desiderata and
- ◇ software design proposals.

- ★ A final principle of *narration* is to achieve something:
- ◇ understanding of what has been described (prescribed, specified),
 - ◇ ability to communicate that understanding to others, and
 - ◇ firm bases for continued, meaningful, productive work



- We refer to a later lecture on models and modelling
- for an extensive discussion of a highly enumerated set
- of properties of models, that is, of the things we narrate.

Techniques 4 *Narrative Documents*:

- A main technique of *narration* is aimed at creating pleasing descriptions (prescriptions and specifications).
- Many techniques in Vol. 1 of this series of textbooks in software engineering are aimed at creating such documents.
 - ★ Abstraction, also when just using informal language, is of utmost importance.
 - ★ Simplicity and low number of concepts are likewise important.
 - ★ Emphasising properties, as for axiomatic presentations, or (mathematical) models represents a choice between techniques.

- ★ Vol. 2 focuses on the following techniques.
- ★ Hierarchical or composite (“top-down” or “bottom-up”) presentations are covered in Vol. 2, Chap. 2.
- ★ Designating phenomena or concepts by denotations or, operationally, by computations, is covered in Vol. 2, Chap. 3.
- ★ Composing models from appropriately chosen state and context configurations is covered in Vol. 2, Chap. 4.
- ★ Expressing concurrent and temporal phenomena is the subject of Chaps. 5, and 12–15 of Vol. 2.
- So, all in all, previous volumes cover many narration techniques

.

■

Tools 2.3 *Narrative Documentation:*

- The tools mentioned in earlier lectures apply equally well here.
 - ★ As narratives are usually large to very large,
 - ★ the tools must be able to handle essentially
 - ★ indefinitely large narratives

Formal Descriptions

- Other lecture series cover formal specification.

Characterisation 2.67 • By *formalisation* we mean a specification (description, prescription) expressed in a formal language.

- A *formal specification language* is one which has
 - ★ a precise, mathematical syntax,
 - ★ a precise mathematical semantics and, usually,
 - ★ a mathematical logic proof system congruent with the semantics

.



Pragmatics, Semantics and Syntax of Formalisations

- The reason for, i.e., the *pragmatics* of, formalisation was stated clearly in the preface to Vol. 1. We will summarise salient points here:
 - ★ Formalisation brings a degree of clarity that cannot be obtained by using only natural plus professional (universe of discourse) language narratives.
 - ★ Formalisation allows us to reason precisely about consistency and completeness of the formalised descriptions, and whether these formalisations satisfy otherwise formally expressed properties.

- ★ Formalisation allows us to relate phases, stages and steps of development, including proving correctness of development.
- ★ In other words, prove that a concrete stage of development is a correct implementation of a previous, more abstract stage of development.
- The *semantics* and *syntax* of formalisations are, as the prefix *formal* indicates, precise, and they differ from formal specification language to language.

On Formalisation

- Those course students who study the present volume in its informal version may skip the next paragraphs — as they can always skip the specially framed *formal version* texts.
- The extensive coverage of numerous formalisation principles, techniques and tools in earlier volumes does not require us to elaborate further on formalisation other than the summary given next.

- A formal specification, when expressed in **RSL**, typically contains the following parts:
 - ★ **Definition of types**, i.e., of spaces of values, whether abstractly, in terms of sorts (just named types), or concretely, say in terms of integer, natural number, real, Boolean, set, Cartesian, list, map or function spaces.
 - ★ If types are defined as sorts, then a specification would typically contain a number of **function signatures** of functions that observe properties of sort values and which generate such values (i.e., *observers* and *generators*), as well as
 - ★ **axioms** over functions and values.

- ★ **Definition of functions** (i.e., of function values), where the form of these definitions may vary from explicit function value definitions, via pre/post condition specified function values, to fully algebraically (i.e., axiomatically) specified function values.
- ★ Some formal specifications may be expressed imperatively, in which case the formal specification will additionally contain a **declaration of variables**.

★ Some formal specifications may be expressed in terms of processes that may be composed from other processes: In parallel, with external or with internal nondeterminism, or interlocked. These processes synchronise and communicate with one another over **declared channels**.

- These parts may be grouped into schemes, classes and objects, such as covered in Vol. 2, Chap.10.

Example 2.37 *Formalising a Document Domain*: We follow up on Example 2.36 by suggesting a formalisation.

type $D, O, V, C, L, T, \text{Txt}$ $\text{FWD}, \text{REV} = \text{Txt} \rightarrow \text{Txt}$ $\text{EDIT} = \text{FWD} \times \text{REV}$ **axiom** $\forall (f,r):\text{EDIT} \cdot \forall d:D \cdot r(f(d))=d=f(r(d))$ **value** $\text{is_O}: D \rightarrow \mathbf{Bool}$ $\text{is_V}: D \rightarrow \mathbf{Bool}$ $\text{is_C}: D \rightarrow \mathbf{Bool}$ $\text{create}: \text{Txt} \times L \times T \rightarrow O$ $\text{edit}: \text{EDIT} \times L \times T \rightarrow D \rightarrow V$ $\text{copy}: L \times T \rightarrow D \rightarrow C$ $\text{trace}: D \rightarrow D^*$ **axiom ...**

Methodological Consequences: Principles, Techniques and Tools

Principles 2.7 • The principle of *being formal* rests on a number of assumptions:

- ★ that it is possible to formalise and
- ★ that it brings benefit to formalise:
 - ◇ that formalisations can be communicated
 - ◇ and that formalisations can be used in further development, for verification and for theory formation.

- Once these tenets hold, we can apply the principles of *formalisation*:
 - ★ Formalise, but do not over-formalise.
 - ★ Choose an appropriate abstraction level.
 - ★ Formalise what is beneficial to formalise:
 - ◇ where the formalisation can be communicated and
 - ◇ can serve as a basis for further development

•

Techniques 5 ● Volumes 1 and 2 cover many techniques of *formalisation*.

- Hence we shall not try here to boil the many subprinciples and subtechniques of formalisation down to a single paragraph
-

Tools 2.4 *Formalisation Documentation:*

- The tools mentioned in an earlier lecture apply here.
- But, in addition, quite sophisticated software packages are needed as tool support for formalisation.
- Each formal specification language usually comes with its own more or less complete set of tools:
 - ★ syntax tools: editors, type checkers, and so on;
 - ★ analysis tools: theorem provers or proof assistants, model checkers, testing tools, and so on; and
 - ★ output tools: symbolic interpreters, compilers, and so on.
- For the documentation of formal models only the syntax tools are necessary

.

■

Discussion of Descriptive Documentation

- Description is an art.
- Much can, however, be learned.
- Next lectures will present
 - ★ principles, techniques and tools
 - ★ describing phenomena and concepts, by means of
 - ★ designations, definitions and refutable assertions.

- That is,
 - ★ although we shall say no more in the present lecture on “how to describe”,
 - ★ we shall present much more material on this crucial subject in the next lectures.
- Volumes 1 and 2 enunciate many principles and techniques for description (prescription, specification).

Topic 15

Analytic Document Parts

- We remind the student of our earlier characterisation of the concept of analysis:

Characterisation 2.68 By an *analysis* we mean

- a text which proves (i.e., reasons over),
- or designates, properties of some other text,
- or properties that are claimed to hold between pairs of texts

.



- Analysis documents may be informal or formal.
- Formal analysis documents can only be achieved if the documents being analysed are themselves formal.

Pragmatics, Semantics and Syntax

- The *pragmatics* of some analysis document, i.e., our reason for having analysis documents (or document texts), is that we wish to have a certain degree of trust in the single or paired documents that are being analysed.
- The *semantics* of formal analysis documents is typically that of a proof, or a model check.
- The *syntax* of formal analysis documents is closely tied to the proof system or the model checking system of the formal specification language being used (i.e., proof rules, model checking property specification languages and their computerised support).

Categories of Analytical Document Parts

- We shall consider four kinds of analytical document parts:
 - ★ *concept formation* documents
 - ★ *validation*
 - ★ *verification, model check* and *test* documents, and
 - ★ *theory formation* documents

- Analysing texts — for any of the purposes implied by the four kinds of analytical document parts — is a hallmark of software development.
- In other engineering branches the developer, the engineer, sets up, typically differential equations, as models of what is being developed. Analyses these, including performing calculations and even computations over them.

- In software development, mathematical, model-oriented or algebraic and logic specifications take the place of the civil, or mechanical, or aeronautics engineers' differential equations.
- And mathematical, algebraic and logical reasoning, i.e., analysis, can be objectively carried out.
- In these lectures we shall (“officially”) not deploy any formalisms,
- hence our analyses will be informal.
- In this, we may be said to not live up to the “hallmark”.

Concept Formation

Characterisation 2.69 By *concept formation* — in the context of analytic software development documentation — we shall understand

- the creation,
- as the result of a study, typically of rough sketches,
- of a number of concepts, i.e., abstract notions
- of otherwise described phenomena, or other concepts

.



- Analysis is partly an art.
- It takes training and experience to do it well.
- It involves abstraction.
- Concept formation is likewise an art.
- Only by being shown many examples can most students and practitioners of software engineering learn to form abstract, pleasing concepts.

Example 2.38 *Rough Sketch Analysis and Formation of Concepts:* We analyse the rough sketch freight logistics of Example 2.34.

- There are the concrete phenomena of trucks, freight trains, air cargo aircraft and line or bulk carrier boats.
- We abstract these into one concept: conveyors.
- There are the concrete phenomena of trucking and freight train depots, of airports and of harbours.
- We abstract these into one concept: hubs.
- There are the notions of roads, rail lines, air traffic corridors, and sea lanes.
- We abstract these into one concept: routes.



Pragmatics, Semantics and Syntax

- The purpose, i.e., the *pragmatics*, of concept formation
 - ★ is to base the narrative and its formalisation on the abstracted, usually simplifying as well as unifying concepts —
 - ★ rather than on a variety of near-similar concrete phenomena.
- The concept formation document texts provide the link between the two levels of abstraction.

- The *semantics* of concept formation
 - ★ is really one of simplification
 - ★ through generalisation
 - ★ and substitution.
- The *syntax* of concept formation
 - ★ is not prescribed, but could amount to
 - ★ a series of pairs of texts:
 - ★ The (concrete usually manifest phenomenon designating) text to be generalised,
 - ★ and the abstract conceptual text serving as substitution.

Validation

Characterisation 2.70 By *validation* — in the context of analytic software development documentation — we shall understand

- a process, and the resulting (analytic) documents,
- in which some descriptive (prescriptive, or specification) documents are being inspected by stakeholders of the relevant universe of discourse, and
- in which whatever is being described (prescribed, specified) is being positively and/or negatively reviewed (i.e., positively and/or negatively criticised) by these,
- including the pointing out, if necessary, of inconsistencies, incompletenesses, conflicts and errors of description

.



- We will deal with validation extensively in later lectures on domain and requirements validation.
- That is: Validation (including its documentation) will take on a rather special role in these lectures.

Verification, Model Checking, Testing

Characterisation 2.71 By *verification*, *model checking* and *testing* — in the context of analytic software development documentation — we shall understand

- a process, and the resulting (analytic) documents,
- in which some descriptive (prescriptive, or specification) documents are being studied
- in order to ascertain whether what is described (prescribed or specified) in the analysed descriptive documents
- satisfies certain (claimed or otherwise expected) properties

.



- Verification, model checking and testing is typically concerned with such things as:
 - ★ Is a domain requirements prescription properly related to a claimed, underlying domain description?
 - ★ Does a software design specification represent a correct implementation of its requirements prescription, given the assumptions expressed in a related domain description?

- Verification and model checking are essentially only meaningful if the various descriptions, prescriptions and specifications are expressed with a suitable mathematical rigour.
- Testing is only meaningful if the various descriptions, prescriptions and specifications can form the direct basis for computer executions (as can code).
- We earlier made statements as to why these volumes do not cover formal verification to any noticeable extent.

Theory Formation

Characterisation 2.72 By *theory formation* — in the context of analytic software development documentation — we shall understand

- a process and the resulting (analytic) documents, i.e.,
- a study of some described universe of discourse
- with the objective of, and resulting in,
- the discovery (i.e., formulation), respectively proof (i.e., verification), of properties of the model, i.e., of the descriptions (prescriptions, resp. specifications)

- We shall not bring in theory formation examples,
- but we shall hint at domain theory formation in later lectures.

Discussion of Analytic Documentation

General

- We have rather briefly surveyed the notion of analysis and analytic documentation.
- Since we do not bring in substantial material on formal verification, model checking or testing,
 - ★ we shall basically only be illustrating some concept formation.
 - ★ Domain and requirements validation is rather thoroughly covered in later lectures (on domain, respectively requirements validation).

Methodological Consequences: Principles, Techniques and Tools

Principle 2.8 *Analysis*:

- Hand in hand with description (prescription, specification),
 - ★ the developer, ideally, reasons about
 - ★ what the descriptive (prescriptive, specifying) documents
 - ★ are describing (prescribing, specifying).

- The developer does so in either of a number of ways:
 - ★ By systematic, but informal reasoning;
 - ★ or by rigorous reasoning, stating proof obligations, formulating lemmas and theorems;
 - ★ or by formal reasoning,
 - ◇ formally verifying stated lemmas, etc.,
 - ◇ model checking stated assertions,
 - ◇ or testing these rigorously

Principles 2.9 • *Analysis documents* must be convincing,

- definitive and
- final

.



- Since we do not bring in any substantial material on analysis,
- we shall refrain from stating techniques and tools
- for the construction of analysis documents.

Topic 16

Discussion

General

- *Documentation is almost all!*
- In the introduction to this lecture topic on documentation we claimed *documentation is all!*
- So what is the difference?

- The difference is this:
 - ★ In order to document domains and requirements, we must acquire domain knowledge, respectively requirements expectations.
 - ★ And in order to come up with pleasing models (descriptions, prescriptions, and specifications), we must analyse.
- So *documentation is all* if we include the documentation of the acquisition and the analysis efforts — as we shall indeed argue should be done.
- Such arguments will be put forward in much later lectures.

Summary of Lecture

This lecture has conveyed the following:

- When developing, hence documenting software, the following documentation must be constructed:
 - ★ (i) informative documents,
 - ★ (ii) descriptive documents, and
 - ★ (iii) analytic documents.
- Figure 2.6 shows a structure of the various kinds of documentation relevant for any stage of development.



Figure 2.6: A graphic overview of stage documentation

As concerns

- (i) informative documents, these are the kinds of possibly relevant document parts:
 - ★ (i.1) partners: clients and developer document parts;
 - ★ (i.2) current situation, needs, ideas and concepts document parts;
 - ★ (i.3) scope, span and synopsis document parts;
 - ★ (i.4) assumptions and dependencies + implicit/derivative goals document parts;
 - ★ (i.5) contract and design brief documents; and
 - ★ (i.6) logbook.


As concerns

- (ii) descriptive documents, these are the kinds of possibly relevant document parts:
 - ★ (ii.1) rough sketches,
 - ★ (ii.2) terminologies,
 - ★ (ii.3) narratives and, ideally,
 - ★ (ii.4) formalisations (although such will only be hinted at in this volume).

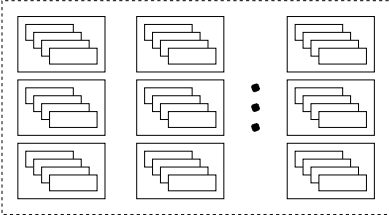
And as concerns

- (iii) analytic documents, these are the kinds of possibly relevant document parts:
 - ★ (iii.1) concept formation,
 - ★ (iii.2) validation,
 - ★ (iii.3) verification, model checking, testing and
 - ★ (iii.4) theory formation.

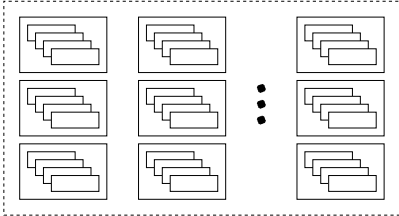
- Figure 2.7 shows a structure of the various kinds of documentation relevant for an entire development.
- Please note our careful use of the two terms:
 - ★ documentation of one kind or another, but not mixed.
 - ★ And hence, document as such a suitable “mix” of documentation.

SOFTWARE ENGINEERING: Domains, Requirements, and Software Design	Nancy Lectures	Volume 3	© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark	and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany	
2.4 Discussion	Topic: 16, Foil: 8/365				

Domain Phase Stages of Development



Requirements Phase Stages of Development



Software Design Phase Stages of Development

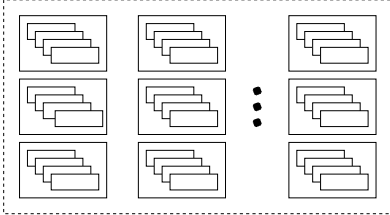


Figure 2.7: A graphic overview of full development documentation

- The purpose of showing Fig. 2.6 is to structure your awareness of the multitude of document kinds.
- The purpose of showing Fig. 2.7 is to structure your awareness of the multitude of documents.
 - ★ In any typical development there will, indeed, be very many documents,
 - ★ and it is therefore of utmost importance
 - ◇ to keep track of these documents,
 - ◇ of their many versions
 - ◇ and to be able to compose “configurations” of the right versions.

Methodological Consequences: Principles, Techniques and Tools

This series of textbooks on software engineering espouses the following principle of documentation:

Principles 2.10 • *Documentation* is everything.

- *Document everything:*

- ★ all information,
- ★ all descriptions (prescriptions, specifications),
- ★ all analyses.

- For all phases document
 - ★ domain models,
 - ★ requirements models and
 - ★ software designs.
- Maintain all documents,
 - ★ monitor and control all versions, and
 - ★ keep them updated,
- at almost any cost

.

