INF5120
”Modelbased System development”

Lecture 10: 20.03.2017

Anton Landmark

Anton.Landmark@sintef.no

Modelling from Experience

- In this talk **no** criticism is intended, even if I some times may be using verbally “strong colours”.
- From Aviation: ***You do not live long enough to make all mistakes yourself; learn from others***.
- My concept of “experience on modelling” is mainly based on the making of various types of computer based systems, small and large, and then on participating in developing : “development methods and techniques”.
- We will first look into some general modelling concepts and elements (including some illustrations), then we will look at an example of a modelling problem.

What is " Experience"

- "Experience" (to me) is to meet and feel the *real consequences* of the decisions/actions made in the "creation process". (Here: consequences of the decisions made during the modelling/design phases.)
- The only way I see to obtain this experience is to be a responsible member of the teams doing:
  - [Making decisions]: modelling / design work, implementation.
  - [Feel consequences]: introduction to groups of the users, do maintenance work for say a year or two.
  - Then: make an analyses:
    - WHAT (was good and not-so-good) - HOW (should it have been made) - WHY (did parts became not-so-good)
    - Understand why parts of the modelling/design process was faulty/inadequate).

”fritt” etter Piet Hein

Den greske vismann Hr. A. Klit
som sa så mangt, så godt, så tit
Han sa blandt annet Panta Rei, *
det samme si’er til tider jeg.

* Panta Rei: Alt flyter (It's all afloat / a mess)

- Modelling:
- **“Straighten out the mess”:**
  Making ‘order and method’ in the chaos.

Remember: Well defined terminology

- Different persons do not always mean the same thing when using the same word.
- "When I use a word, Humpty Dumpty said in rather a scornful tone, it means just what I choose it to mean - - - neither more nor less."

  (From: Lewis Carroll, Through the Looking Glass, Chapter 6 - Humpty Dumpty.)
- The meaning contained in words are changing over time (and rather fast, due to misunderstandings and neglect!) (Examples: Byte, Type, Data, Information)
  - Information (Rules for encoding /decoding) Data
- Some notions tends to be widely misunderstood, such as 'Real time systems' and 'recursive / reentrant'. (people THINK they understand it, based on misleading naming)

What is “MODELLING”

- “Modelling” means to make a ***presentation*** of ***all*** the ***important characteristic features*** and ***elements*** of ‘some subject’ *for a given* ***purpose*** in a *given* **context**. The Subject: what is being modelled. (a thing, an organisation, an arrangement, some type of a process, etc.)
- This definition contains two equally important elements:
  - *a presentation* (that can as far as possible be understood uniquely). [‘Symbols / lay-out / interaction’]
  - *a set of all important characteristic features and elements*. This set will vary widely from case to case, being fully dependant upon the ‘subject’ being modelled.
    [a set of *items / documents* to be specified /defined.]
  - Let us take a broader view of “Modelling”, just to widen our perspective (**various examples** of modelling).

Modelling example - 1

- Modelling is not something new;
  it has at least been going on for
  the last 22 000 – 24 000 years:
  - Venus from Willendorf
  - Venus_of_Arles
  - Twiggy
- Purpose: "Show the desirable female attributes".

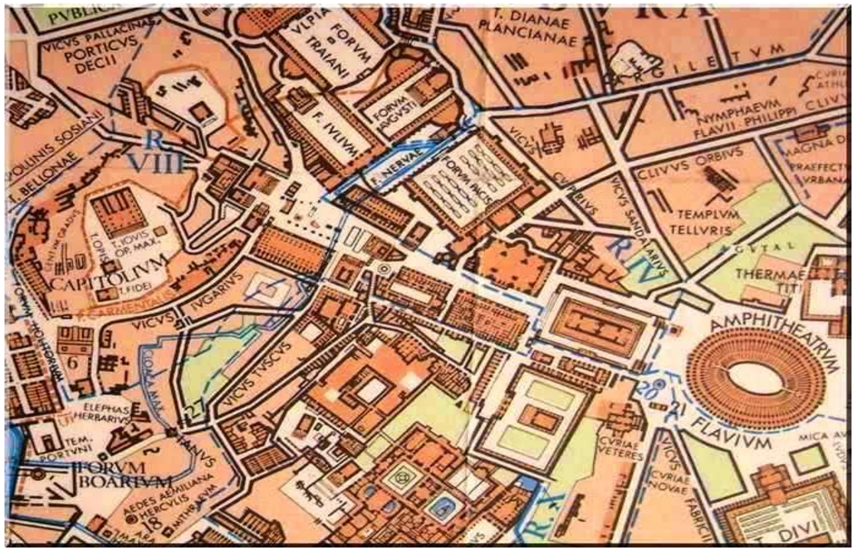
"Whatever is hard to obtain, becomes desirable."
(and "in" / "fashionable").

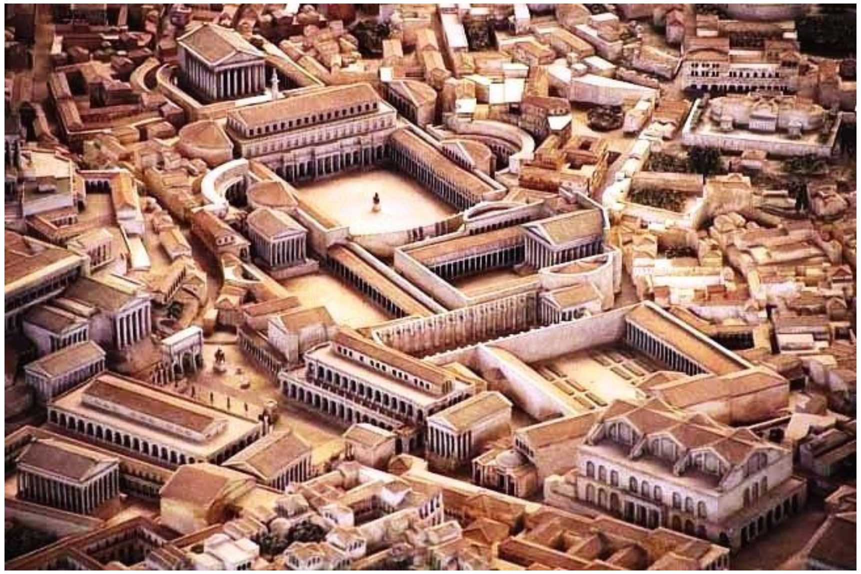
Modelling example - 2

- More modern modelling:
- Modelling for entirely different purposes: For which ones??
  - P – 51 Mustang (Looks and behaviour)
  - Douglas Commercial # 3 (DC-3) (Structure Before covering?)
  - A variety of airliners (in the window of a travel agency: take a trip?)
- **Essence**: Why were these models made?
- Key words:
  - ***purpose*,**
  - ***expectations***

Modelling example-3

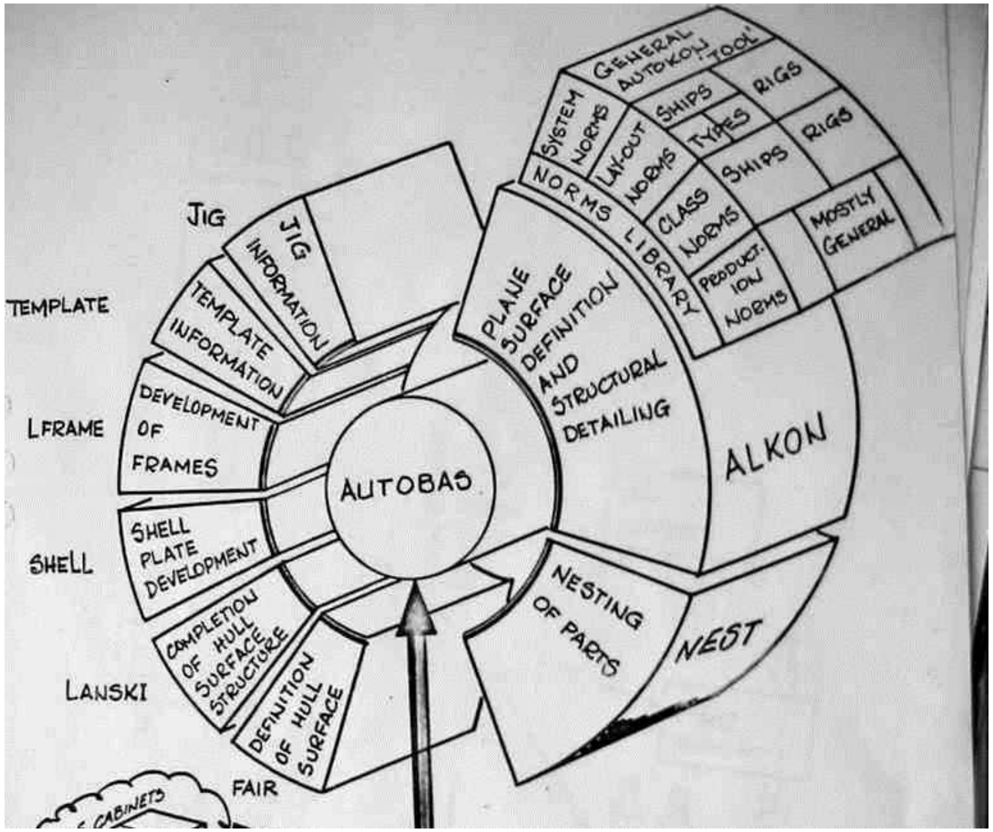
- Modelling for the same and yet for different purposes:
  Map and solid 3D model.
- Both from area in ancient Rome, from ca. 300 - 400 AD. Made for different use/reason.
  - Map: portable, for orientation.
  - Explanatory/visualisation model, stationary.
- Same data? Giving same information?
  Key words:
  - ***Purpose***
  - ***Expectations***





Modelling example - 4

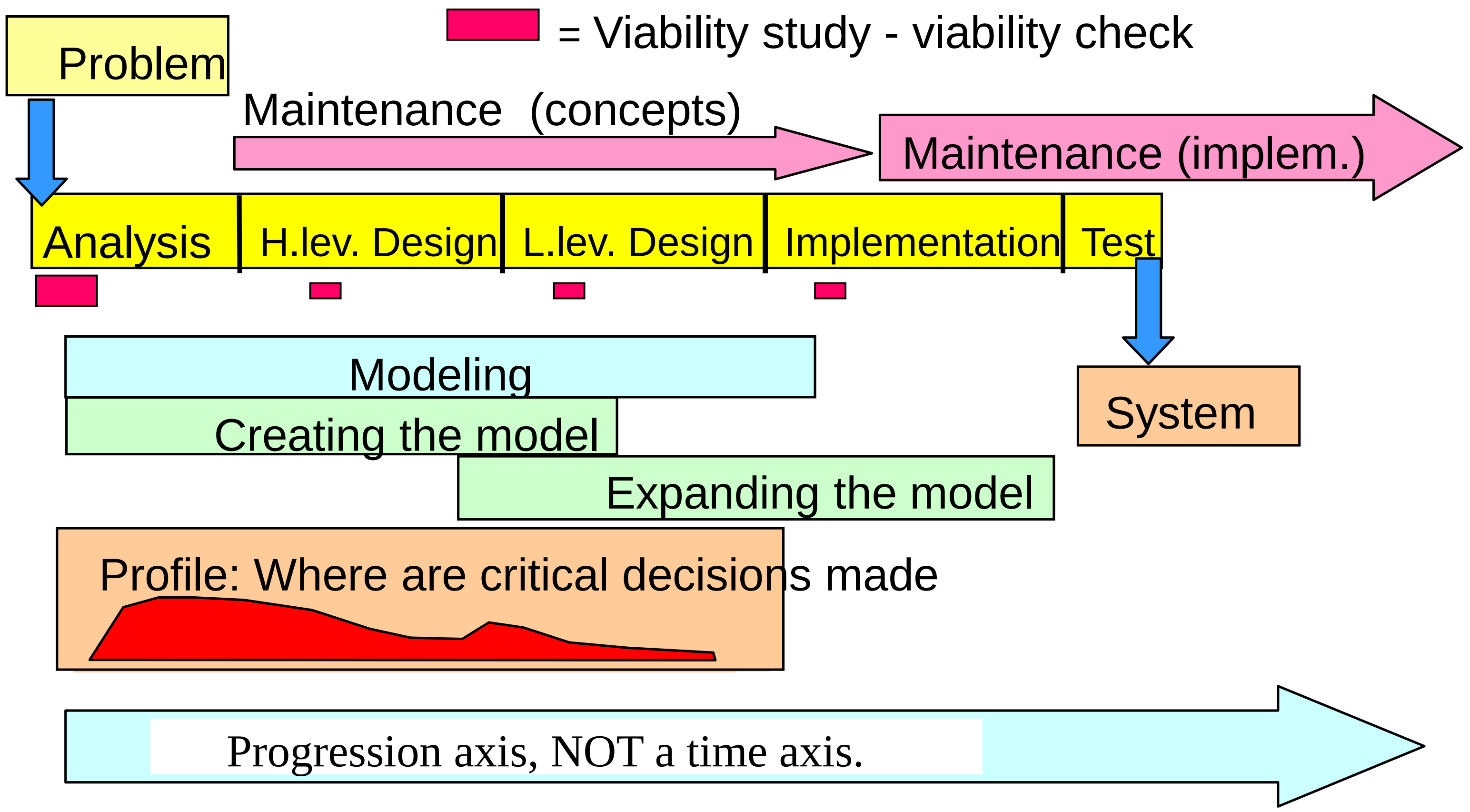
- Part of a high-level model of a computer-based CAD-CAM system



“Modelling”- a part of system development

- In spite of what many people like to believe, the modelling task tends to be a ***comprehensive and critical part*** of the ***system design***.
  - If you possibly want to add some ‘**design**’ phase, it will normally be an *elaboration* of the model elements, not a *replacement* of some of parts (if the model is reasonable good).

System Development Elements



“Modelling”- part of system development (2)

- During the modelling process a lot of decisions are being made.
  - ***These decisions must be the right ones***, based on insight and understanding of the Universe of Discourse (UoD), not just pulled from a hat. (They may have to last for the entire life span of the system.)
  - Some Use cases is not good enough!
- Experience stresses the importance of ***understanding*** the subject for the modelling during the modelling process.
  - And by the understanding, detect and include in the model **<u>all</u>** the important ***characteristic features*** in the subject, including all the ***requirements*** from it.

“Wise men”:

- C. A. R. Hoare:

There are two ways of constructing a software design:

- One way is to make it so simple that there are obviously no deficiencies
- and the other way is to make it so complicated that there are no obvious deficiencies.

- Albert Einstein:

Make things as ***simple as possible***, ***but no simpler***.

(and where is the limit??)

"K I S S ":

"Keep it simple and stupid":

- If you are able to decompose a complex and difficult problem into separate sub-elements in such a way that it all looks "stupidly simple", and yet turns out to be correct, then you have ***really succeeded***:
  - Succeeded in understanding
  - Succeeded in expressing

– Remember Albert Einstein:

Make things as ***simple as possible***, ***<u>but no simpler</u>***.

(and where is the limit: ok simple – too simple??)

The start of a modelling job: Organize it!

- Make a model of the job at hand: Design / organize the modelling team:
  - The team entities/members, responsibility distribution, tasks required, information flow in the job,
  - The methods and tools to be used (and why just those?).
  - Results to be produced.
- Then start using that model:
  - identify which abstract items are to be present, which entities must possess what understandings and why, carried by which information and in which data representation.

Start the modelling job: make the top level !

- Make a model of the job at hand: Design / organize the <u>modelling itself</u>:
  - Split the system / problem into a set of cooperating task handlers, where each one has a task and a responsibility of its own.
  - The elements cooperate in performing/handling the complete *system* task.
  - They *only* cooperate by sending messages to each other, requesting the receiver to perform some ‘job’ within the receivers realm (and which the sender may thus not perform).

Then: make the next, lower level !

- Make a model of the job at hand: Design / define ***one*** element/handler/component/object (from here on denoted ***object***) from the level above.
  - Only *one* upper-level object per page.
  - Component/object based: Only the object itself may know anything of its internal contents/structure. (not even a component above or below may know anything).
  - Remember: Component/object based: *Encapsulation*.
  - Remember: Any object may only know-of/cooperating with other object within its own realm (within same page). This means: no message/call may be sent to someone at some other page!!!

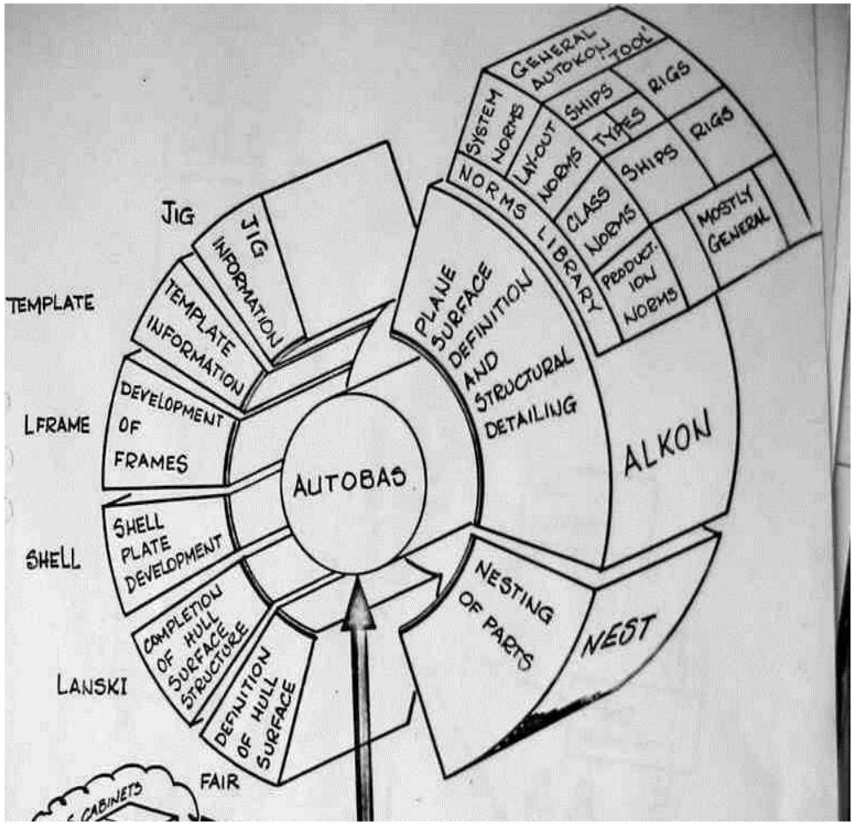
Modelling (1) typical elements

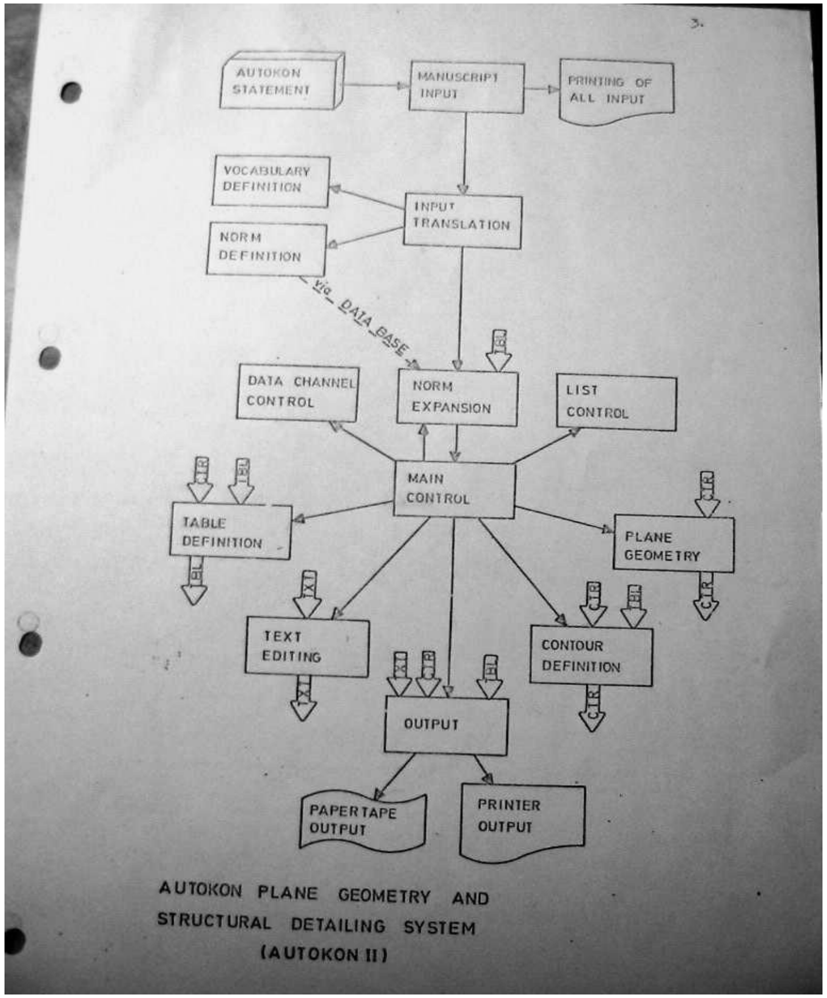
- Modelling may be a complex and a comprehensive job.
- A large set of ‘items / considerations’ are to be considered, many of which are far too often overlooked.
  - Identifying: life span of system, areas of responsibility, building blocks, internal unit collaboration, cost limitations, quality parameters such as:
    - MTBF (stability)
    - TTR (consequences of down time, planned or not)
    - Accuracy (inherent in data, computational)
    - Reliability (system trustworthy?)
    - Value ranges of all quantities, and how are they enforced in the system?

Modelling (2) levels and *modifiable*

- A major part of the modelling task is to 'design' the set of components involved (in as many levels as needed), how they cooperate, (within each level), what sort of elements they handle and all the interaction. (***One level at the time***, please, otherwise chaos!)
- All levels in a top – down structure: Each object/component ***defined on one sheet****! (****readability****), (and* ***no*** *overloading).*
- *Present* functionality is merely one item in the crowd, but receives normally (unduly?) high / much attention.
  - Remember: it is only "*Present*"; it must be *Modifiable* , both as to functionality and language, for any system intended to last more than half a year (or one year at the most).

Example of one-page presentations





Example of low level def.

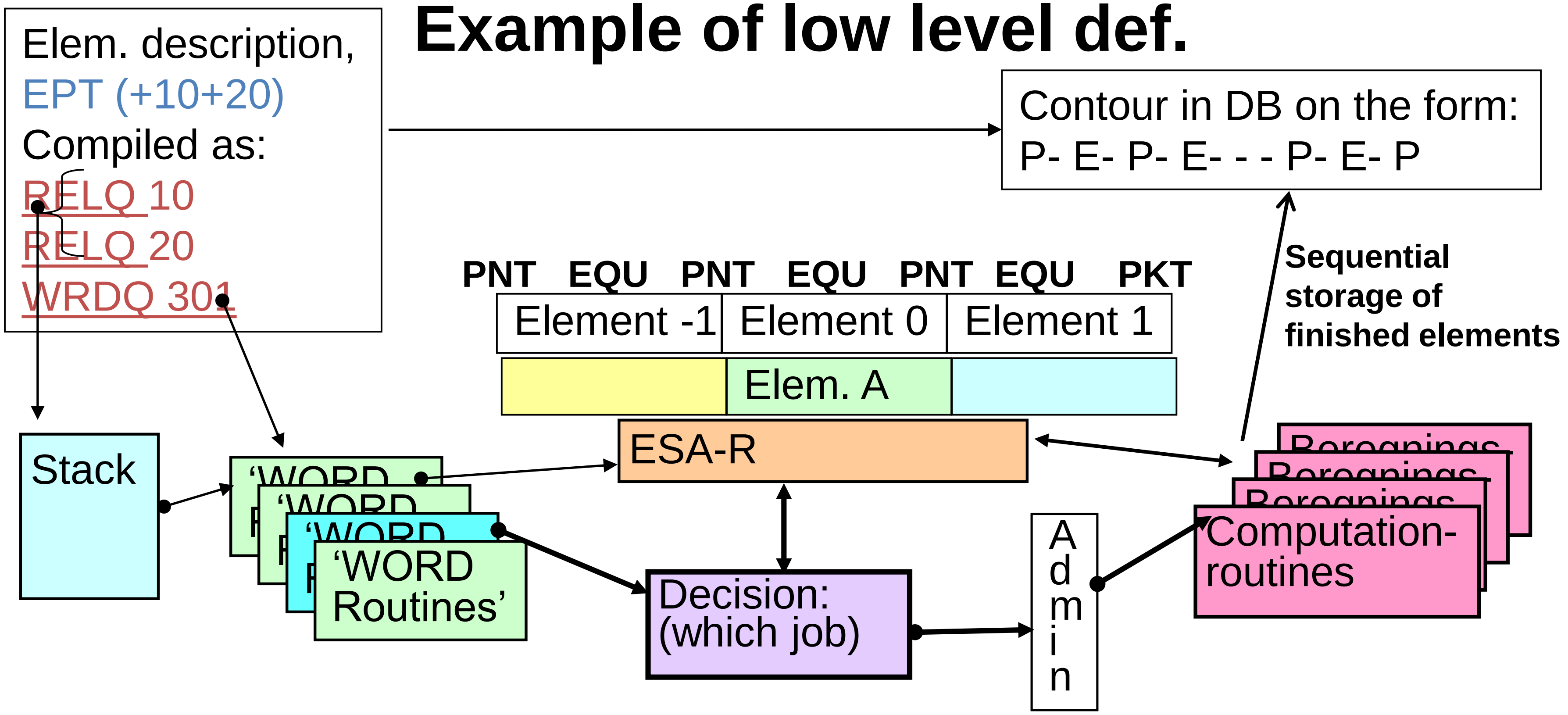


Table-of-contents over current relevant data available (Key)

SL	CIR	SPT	EQU	EPT	PNT-1	PNT-2	PNT-3	.	.	INT	TG	.	Former	SL	CIR	EQU	.	INT	TG	.	Next	SL	CIR	EQU	.	INT	TG	.

SL: TG
CIR: TG

CIR:
CIR:

CIR: TG
CIR: TG

EPT:
Word-ID 301
Group 2
Variant 45
Syntax W23/ N27
Norm 301

Keyword: documentation mandatory

- A model is made for a **purpose**. **Define which**.
  - ***Document*** which one properly and precisely .
- The Purpose will carry with it a **set** *of* ***important characteristic features and requirements***
  - ***Document*** properly and precisely ***all these sets***.
- Without this ***documentation***, no correct and reliable models or design can be made, as it is the ***critical part of the model definition***.
- This documentation is ***the foundation*** for selecting/creating the set of elements constituting the model. (*Representation)*.
- **Un-**documented items are **non-**existent.

Examples of Model Purpose

- The *type of model* to be made and the *modelling process* depends on the type/nature of "***Purpose***", e.g.:
  - **A tutorial** / visualisation, e.g. a map (of area or of economy)
  - **Clarification** / visualisation / testing (of some problem area)?
  - **Development** of some computer based system?
- The two first ones tends normally to be relatively simple to handle.
- The last one is normally the harder one, due to elements like: *abstraction*, expected *life span*, *economy*, the variety of *requirements*, needs for *modifications* / *extensions* etc..
- We will in the following look at some typical *types of models* and types of *modelling processes*.

Types/purposes of Models

- **All modelling is simplifications.**
- ***"keep things as simple as possible, but no simpler".***
  - *Albert Einstein*
- During modelling we focus on specific aspects, see them from a specific angle and in a special light.
- Thus: no model shows it all, you need a set of models for the purpose at hand, normally at several levels.
- Normally our "area of concern" (or "Universe of Discourse", hereafter UoD) may be quite complex.
  - We need "clarification models" in order to "see and understand" .
  - We want to know what UoD consists of, some composition / structure description (e.e. *object models*).
  - We want to know what is going on, the functionality, (*use cases*, *message diagrams*.)
- Based on this, we may make "construction drawings" for an implementation: '*implementation models*'.

Elements inside / outside modelling

- Elements that the *modeller must be aware of* and understand, even if it strictly is not part of the modelling process, such notions that may ***indirectly affect*** the modelling and the model:

  Data Types (operations - data storage),
  Components – Objects
  Modelling aspects including:

  reliability, economy, accuracy.
  Type checking in various languages**?**

- A rich repertoire often signals bad modelling/design: lots of things were missing, the system is heavily patched up, and consequently hardly consistent.

Types (1)

- Type declarations is defining which operations are permitted, and how to execute them.
  - Has also become a definition of storage lay out, (partially caused by uncritical development of various assembler languages, e.g. C, partially by developers being low on experience, creating "hybrids").
  - E.g. ”+” in (Int + Int) and in (Real + Real)
  - They are not the same operations, (even if this is not always understood). (And how about Char + Char?)
  - Int: 234 + 123 = 357 (*last digits* lined up for adding)
  - Reals: 2.5634 + 0.3 = 2.8634 (*decimal points* lined up)
  - How about 'A' + 'B'? Is it 'AB', or 131_{10}, or 15_{16}?
  - [ASCII: 'A' is 65_{10} and 'B' is 66_{10}, 131_{10} is outside ASCII],
  - [Hex(adecimal): A_{16} denotes 10_{10}, B_{16} denotes 11_{10}],
  - (symbols 'A' and 'B' being both a character and a digit)

Types (2)

- "Integer " means a value obeying integer arithmetic rules, e.g. 5, 234, 123456789012345*10^4
- "Real " means a value obeying floating point arithmetic rules, e.g. 5.0, 23.4, 0.0003456, 12.34567890123456 or 123456789012345678.90
- Most modern programming languages do <u>not</u> have types such as “Integer” or “Real”, only very limited subsets (even if they tend to <u>pretend</u> they do have so).
- This misunderstanding of “Types” constitutes today a serious hazard to modelling.

The notion “BYTE”

- The directly addressable unit in memory, or one basic element of data.
  - “BYTE” is often mistaken to be an 8-bit unit.(which is an ***octet***)
- We have seen many types of byte units:
  - 4 bits e.g. early Intel processors 4000-series)
  - 5 bits code: Telex
  - 5, 6, 7, 8 or 9 various (early) IBM machines
  - 6, 7, 8 bits: ASCII
  - 6, 9, 12, 18, 36 bits (e.g. Univac 1100-series)
  - 8, 16 bits: ASCII or ISO characters, Integer
  - 32 bits: Integer or Real today.
- 4 to 12 bits: often called BYTE, 12 to 64 (128) bits: a Word. (12, 16, 18, 20, 24, 30, 32, 36, 40, 48, 64, 128)

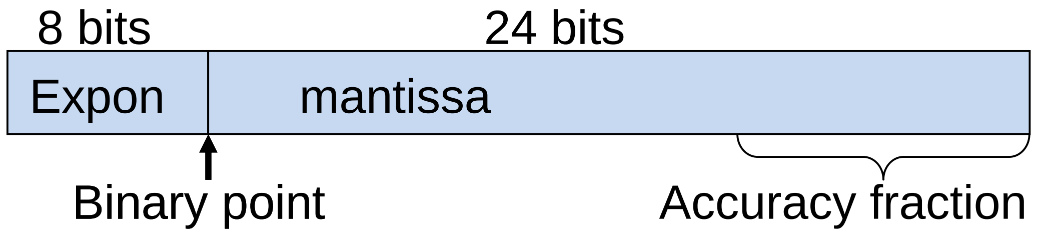
Objects

- An entity, being created, living on the heap for as long as the item it represents exists, then disappears.
  - (different from stack items and “globals”)
- The notion appeared when Televerket wanted to analyse queues of incoming calls at the telephone exchanges (NR, in the 60s.)
- “Individuals coming and going” at some proper time.
- Led to the creation of SIMULA and the notion of objects.

Objects and Components

- The notion of Objects was refined e.g. in Smalltalk.
- 4 ”characteristics” evolved:
  - Encapsulation   (the more volatile one)
  - Polymorphy
  - Inheritance
  - Instantiation
- Unfortunately lack of understanding (and commerce?) led to a mix-up of:
  OO, *OO characteristics* and *OO criteria.*
- OO is a way of **thinking** and **working**, not merely a type of expression and programming.
- **Component**: Closed entity, part of a system, has a responsibility, looks quite like an object, but differs distinctly. (e.g. Inheritance and Instantiation)

Accuracy (1)



- Accuracy fraction:
  - The decimal fraction part of a number is represented as a sum of binary fractions. (e.g. a•1/2+b•1/4+c•1/8+d•1/16+ - - -)
    (when “normalized”: factor a = 1, i.e. bit value is a negated sign bit).
  - 2 significant decimals will require ~12 bits (accuracy fraction), leaving 12 bits for the integer part of the number (4097).
  - With a specified round-off accuracy, ~ 2 more bits are required, leaving 10 bits for the integer part of the number (1023).
- Representing a larger number here will appear to work OK, but accuracy fraction bits are lost, and normally replaced by zeros or sign bits.
- 24 bits can hold up to 16 777 215. (2^{24} -1).
- NB: Underflow or Overflow. (Exponent 7 bits: 0 to 127 or ± 63)

Accuracy (2)

- There are significant differences between Mathematical arithmetic and Computer arithmetic.
  - e.g.: where is “infinity”, and what is “real zero”.
  - e,g,: is $(\sqrt{2})^2$ =2 or $\neq$ 2 (< 2)? ?
  - e.g. : The sequence of factors is significant:
    - (A • B • C) / (D • E • F) will often differ from A/F • B/D • C/E or should it be A/E • B/F • C/D ?
  - A 32 bits Integer can hold max. 2 147 483 677.
  - For Integer used in “Pythagoras” only values less than 46 339 may be used safely. (for Real much less values, 2047 without decimals.)
  - With 2 decimals, a Real should not be greater than ~ 1023 without loosing accuracy.
- ***What must the modelling specify (if anything) in a given situation?*** (situation dependant, must be considered).

Reliability (1)

- Two main aspects:
  - Are all *values correct* (and reliable) at all times, even after a re-start? !
    - Accuracy, correct computations / treatment in all situations, no “foreign” interference (agents, "black-outs”, etc. If no acceptable MTBF is defined: an MTBF ≥ 3 years must be expected!??).
  - *Breakdown* Reliability (Br-rel.):
    - Br-rel. of parts vs. Br-rel of composite units:
    - $(\text{SingPartRelab})^N \geq \text{CompPartRelab}$ for N parts.
    - CompPartRelab is often required ≥ .9995
- *The modelling job* must *specify/device* how to achieve this.

  (You can not ***test*** it into the component, as many seems to believe)

Reliability (2)

- An example of some unreliable software, *giving a wrong answer* and thus costing 97 lives, crash of registrations: N9705C on Sep.29-1959 (fatal. 34) and of N121US on March 17-1960 (fatality 63), one of the very early turbo-prop passenger aircrafts, the Lockheed Electra L-188A.
  - In contrast to modern planes with turbo-engines, the engines were placed in the wing, not underneath it.
  - The wing spars could not pass through the engine, but had to be passed around it, using some tubular structures, prone to resonating at various frequencies. Resonance could/would lead to metal fatigue hazards.
  - A Computer program was made to check for resonance, and tested in the normal way.
  - Alas, it contained an hard to detect error of the following type:

```
IF  ("resonance occurs") GOTO label A  ELSE  GOTO label  B.
Label A :  [ accept the design ]
Label B:   [ discard the design ]
```

- **THE LABELS HAD BEEN SWICHHED AROUND!!!**
- **Actually very hard to detect in normal testing, but quite fatal!**

(the type Electra L-188A is after corrections the basis of the P-3 Orion, a truly reliable workhorse, widely used observation plane, e.g. by USA, UK., Norway etc.)

Maintainability.

- How can a system be maintained and adapted to “current requirements” at any point in its life cycle?
- Keywords:
  - ***Reliable***: How to ensure “no side effects anywhere” when a modification is made? (e.g. in a process or in data element lay-outs)
    - No breach of encapsulation or of responsibility borders.
    - “All treatment of data are done locally”.
  - Economy: what are the cost limits for the maintenance throughout the life expectancy of the system?
    (often tied to ROI.)
- **What must the modelling prescribe to achieve proper maintainability**?

Testing?

- It is a “very hard-to-kill” myth that it is possible to make a system correct and reliable by the use of testing.
- Systematic, comprehensive testing is required to show that every “part” is present: modules and functionality.
- Systematic, comprehensive testing does **<u>not</u>** show the **absence** of bugs.
  - A systematic, comprehensive validation test showing the **absence** of bugs in a “normal-size” system will take something in the order of 10^7 years, running 1 test pr. sec., 24 hours a day, 360 days a year. (and who cares then . . .).
- One of the early Dutch gurus once said: the real trouble is not to get the bugs out, it is to prevent them from ever getting in.
- <u>Correctness / reliability can only be ***modelled*** into a system, not tested into it..</u>

Types of Modelling processes

- ***"keep things as simple as possible, but no simpler".***
  - *Albert Einstein*
- Type of modelling: I choose to include here 3 main types of models, even if several hybrid types often may appear.
  - Data modelling.
  - Functional modelling.
  - Responsibility centred modelling. (~ Role / OO – modelling)
- How about OO? It is a way of thinking/working/documenting, a paradigm, not a way of deciding what the important aspects for a system are.
- Each type of modelling has its own special qualities, good and bad.
  (No single type is suited to handle all sorts of cases.)
- This type of modelling is mainly focusing on what are the important aspects inside the system, not as seen from the outside (the users point of view).

The ‘Data Modelling’ process

- This type of modelling is mainly based on which *data elements* are to be received, handled, stored and reported.
- Characteristics:
  - Very *concrete*, easily *understandable*, one may often think and work in tabular formats.
  - “Hard to modify”. Thus the most vulnerable, primitive, ‘monolithic’ and static type of modelling.
  - Best suited for smaller systems of very limited lifespan, and for some of the simpler real-time applications (one-shot systems). *(i.e. “Systems not to be modified”)*
- Modifications and extensions tends to be rather hazardous, costly and time consuming, due to “all definitions being available everywhere”, lack of strict modularisation, (i.e. no limitations on who is dependent on what data/information).

‘Functional Modelling’ process

- This type of modelling is mainly focusing on data handling and various types of data *transformation / manipulation*.
- Characteristics:
  – May in some cases look a bit similar to Responsibility centred modelling, and may also be combined with this in modular systems.
  – Mathematical libraries offered by many compilers etc. are good examples.
  – Not very useful all by itself, but may be used and combined with other modelling techniques.
  – Used to localize logic of making conclusions/proper transformations / predict consequences or to build standard libraries for handling of complex standard structures within an application.

‘Responsibility centred’ modelling process

- A system consists of a set of subsystems, each one having its own separate and unique “*area of responsibility*.
- Characteristics:
  - A typically modular method, often close to OO thinking.
  - For a module it means: I am the one and the only one here who can and may handle items within this area of responsibility and/or of this type.
  - True modularity implies that any one component may be replaced with another one with same responsibility, with no other modifications being done in the system.
  - Example: a system containing descriptions of some type of contours. The contour representation must for some reason be modified. The contour handling module is replaced, no one else is affected.
  - Being truly modular, it permits a mix of modelling types to be applied among its modules.
  - Strictly limiting: “who knows about what” – “no one sees it all”
- **Gives the most stable, modifiable and maintainable systems of some size.**

Data storage issues (Data bases)

- A traditional relational DB may often be well suited for smaller/ simpler/ trivial systems. ("Keep things as simple *as possible*, but *no simpler*").
- "But no simpler", [**what is needed?**]: larger, more complex, modifiable systems may require specialized storage systems, e.g. for dynamic data, temporary work storage, dynamic overflow-areas etc., including dynamic memory management, possibly with automatic restart capabilities. (e.g. arrays being indexed, stacks having both LIFO and FIFO capability, or dynamically defined name structure, or data formats.)
- For systems using data storage, the data storage unit is an important part, ***and must normally be part of the model***. (In some cases it may be a rather complex unit, requiring skilful modelling).
- For 'trivial systems', standard commercial DB-systems may often suffice.
- For 'non-trivial systems', standard DB-systems are often insufficient.

On Distribution

- Distributed systems tends to require *strictly modular design*.
- Some of the modules may reside on remote locations, thus denying direct access to the software for modifications.
- Some modules may reside on remote locations, being part of other systems, thus denying access for modifications.
- Some modules may reside locally, but belonging to a third party, thus denying access for modifications. (e.g. a commercial DB handler).
- Experience stresses the importance of keeping interfaces intact/stable over time (except strict extensions).
  - If modification of an entry point is required, add a new, extended modified version with new names for the modified entry points.

Authorities etc.

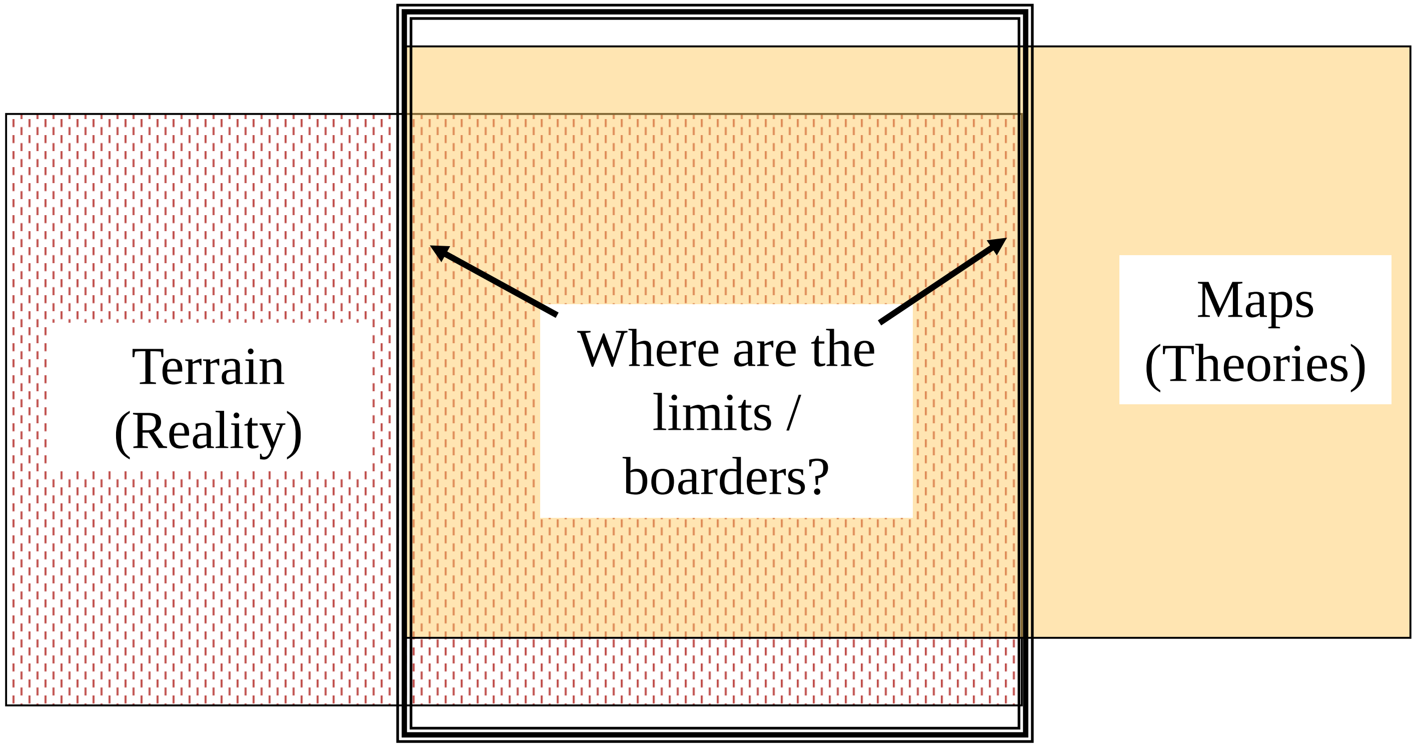
- Remember history: ”the earth is flat, everyone can see that!”
  - The power of authorities;: right or wrong hypotheses.
  - Today we smile a little at this: we know better.
  - But do we not often behave in a way unpleasantly similar to what the old authorities said? Who are verifying this??
  - ”It is written here - - -”: is the written word an authority? ***How about text-books*?!?** (Be critical).
- ”Theology”: do not forget: ”solid foundation of faith”, or: ”my mind is made up, do not confuse me with facts”.
- Some critical attitude may be rather healthy.

How do we work??

- Remember: ”the earth is flat, everyone can see that!”
  - verification?!!
- When the map and the terrain is not in agreement, which one ***is*** correct / incorrect??
- Theory and praxis: which one is the important one?
  - ***Both***! One without the other is nonsense (disaster)
- Parallel:
  - It is important to ***check*** that map and terrain is in agreement, not only assume or believe it. (Meaning that both theoretical and practical aspects are duly taken care of!)
    (Otherwise you will pretty soon be out in ‘the wrong field’)

Reality and theory

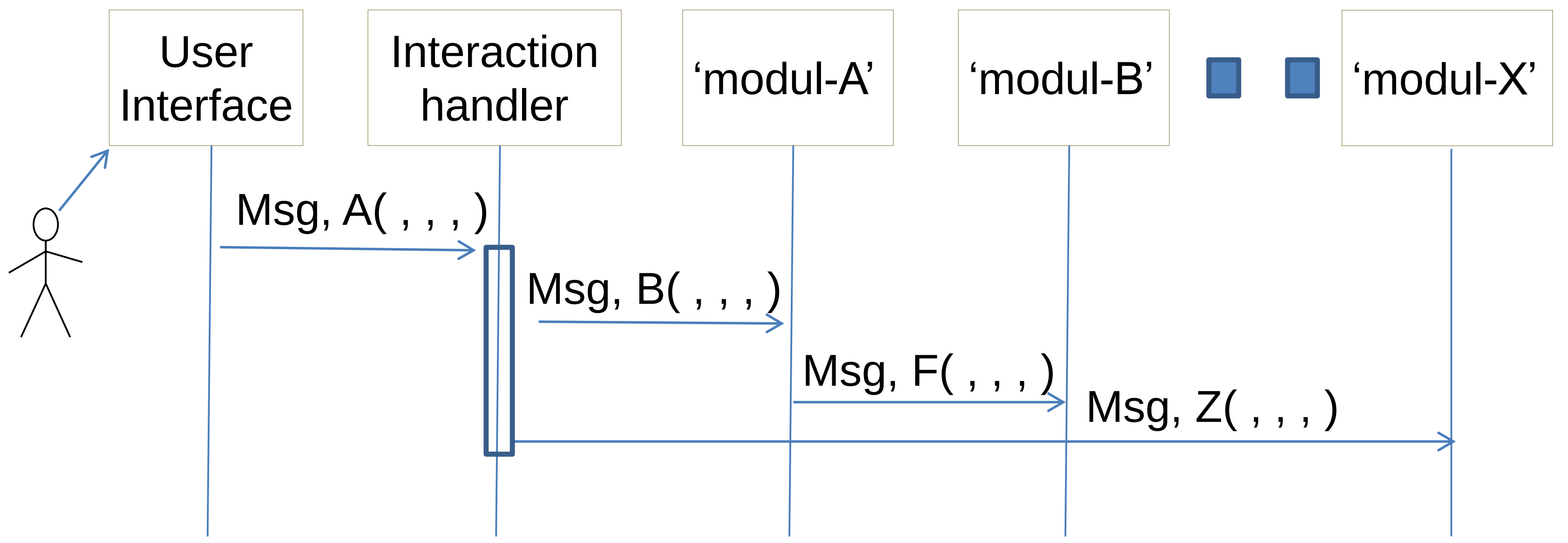
- Combined Reality and theory . . . .



Documentation tools and methods

- The important question here is: in the given modelling job:
  which documents are to be made.
  (decision made at an early phase in the process, see earlier picture)
- Some documents will be text documents, some various types of graphs (saying more than 1000 words does)
- Some diagrams may be: "consists of", some may be: "collaboration diagrams" or "message passing diagrams".
- As to tools: some text editor, some drawing/drafting tools, some 'method specific' tools, e.g. UML.
- UML appears well suited for many 'data modelling' tasks, requiring some extra caution when specifying 'message signatures'. (Developed by some data modellers, trying to give the tool some OO-like appearance.)

Example: Message passing diagram.



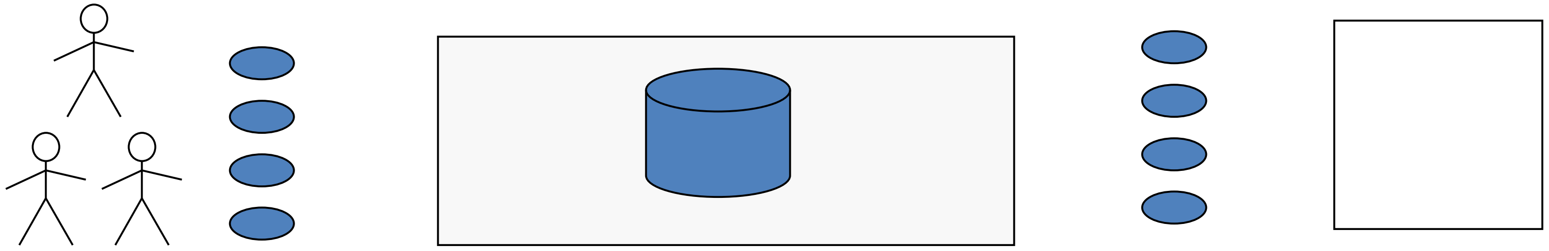
A user is giving a command.
Result: a cascade of messages between modules in order to perform the job asked for. (each message will in due turn return an answer)

A collaboration diagram: the set of messages involved in ' a job', defining the collaboration involved.

”Use-Cases” important & dangerous

- Very important: distinguish clearly between symptoms and problems / requirements!!!
- ”Use-Cases” being important to users, may often be dangerous / misleading to modellers / system developers!
- They may be important input to /output from modellers.
- May not be used in modelling without *uttermost caution*!

  (being only arbitrary samples of system interaction)



Problem example of a system: cold storages at a hospital (1)

- Hospitals may need a set of cold storages, at different temperature ranges, e.g.:
  - Non-freezing storage: between + 4 C and +1 C
  - Freezing storage: between - 20 C and -24 C
  - Freezing storage: between - 80 C and -84 C
  - Freezing storage: between - 180 C and -184 C
- Electric lighting is required several places in connection with these:
  - In rooms housing cold storage units.
  - Inside each cold storage unit.
- Alarm systems:
  - Some temperature is out of range
  - Some person locked in somewhere, due to some error.

Problem example of a system: cold storages at a hospital (2)

- The safe operation of hospital cold storage facilities may be important to patients. Reliable operation has thus high priority.
- Most storages are dependant on electric power, both for control, lighting and for cooling power.
- Some storage facilities are whole rooms, some are units in (special) rooms.
- All electricity are provided through sets of circuits; all of which are provided with proper circuit breakers (fuses).
  - Equipment may (over time) be moved to a different location and thus connected to a different fuse. Start-up current for equipment may be rather high: Units must be prevented from breaking fuses by multiple simultaneous starting,
  - Each circuit has its own fuses. Groups of circuit fuses has a common master fuse.

Problem example of a system: cold storages at a hospital (3)

- The safe operation of hospital cold storage facilities may be important to patients (may even mean life or death). Reliable operation has thus high priority.
- Electric circuits and fuses are of different categories:
  - Supply of cooling Power
  - For control circuitry
  - For alarm circuitry
  - Are there even some alarm circuits watching critical alarms?
- These supplies must be fed by different routs, so e.g. various alarms are not disabled by dropouts in other places. (no cross-over effects.)
- How is a failure in an alarm circuit being detected?
- Some alarms are of a real-time nature. How to ensure proper response?
- Fail safe. When something start failing, dependant processes/aspects might be shut down in a safe and orderly way, e.g. to prevent restart problems.

Problem example of a system: cold storages at a hospital (4)

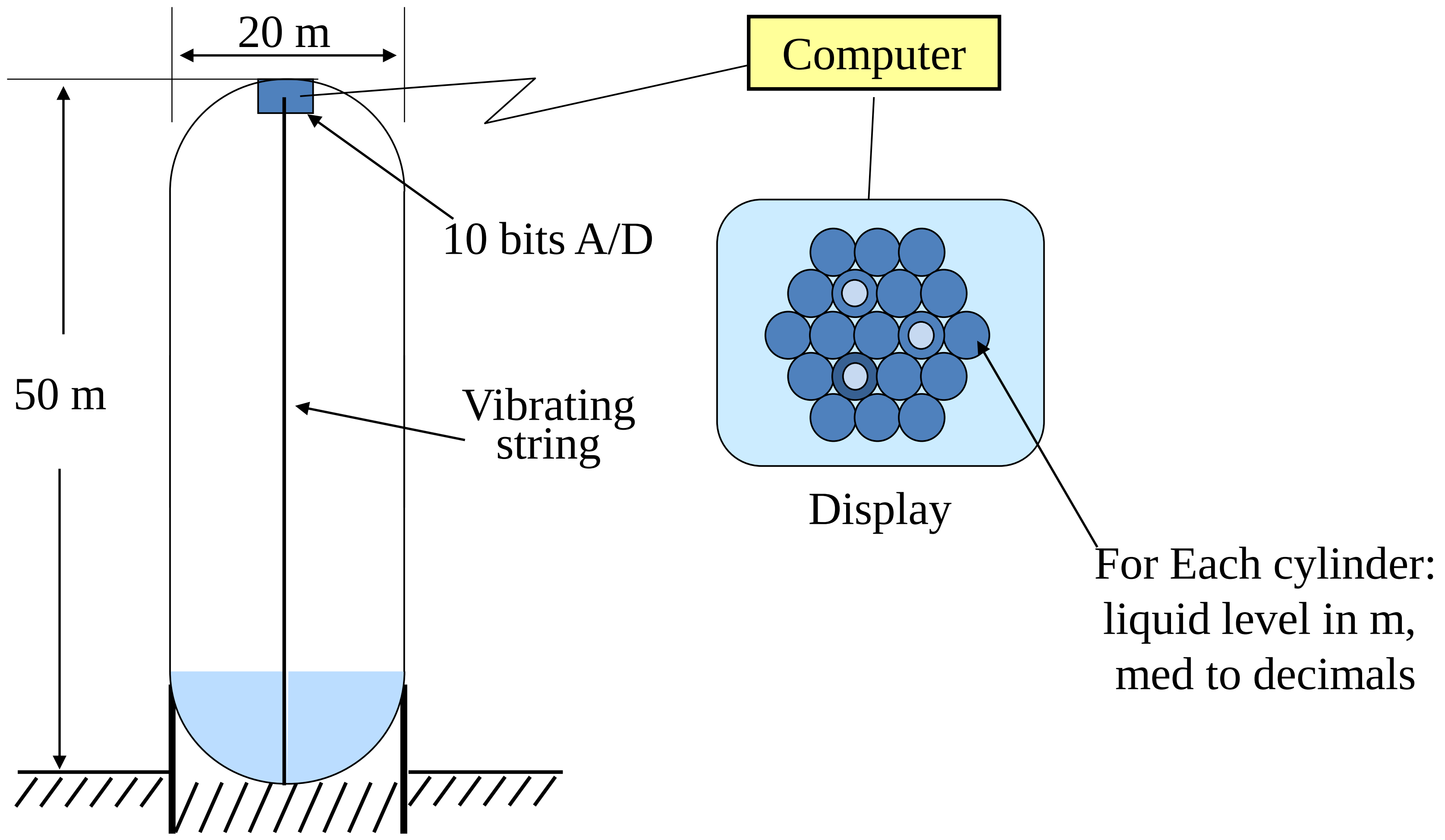
- We have here some of the general requirements. In order to elaborate further on these specifications, we need a more specific hospital with precise and concrete requirements.
- This type of a problem calls for a rather small and concrete model, handling a set of rather simple and physically concrete handlers.
  An *Object-oriented* approach would probably be a good choice.
- We can, however, see that we will need cooperating objects of some classes: Cooling devices, Sensors, Error handlers, Error receivers, Controllers. We may also see that we may need some sub-classes of Cooling devices, for the different temperature ranges (due to polymorphic handling of incoming error reaction messages) and of some other classes.
- How about a set of classes, partly in a hierarchy and a class model?
- An instance object model might then be a good start for modelling.

Problem example of a system: cold storages at a hospital (5)

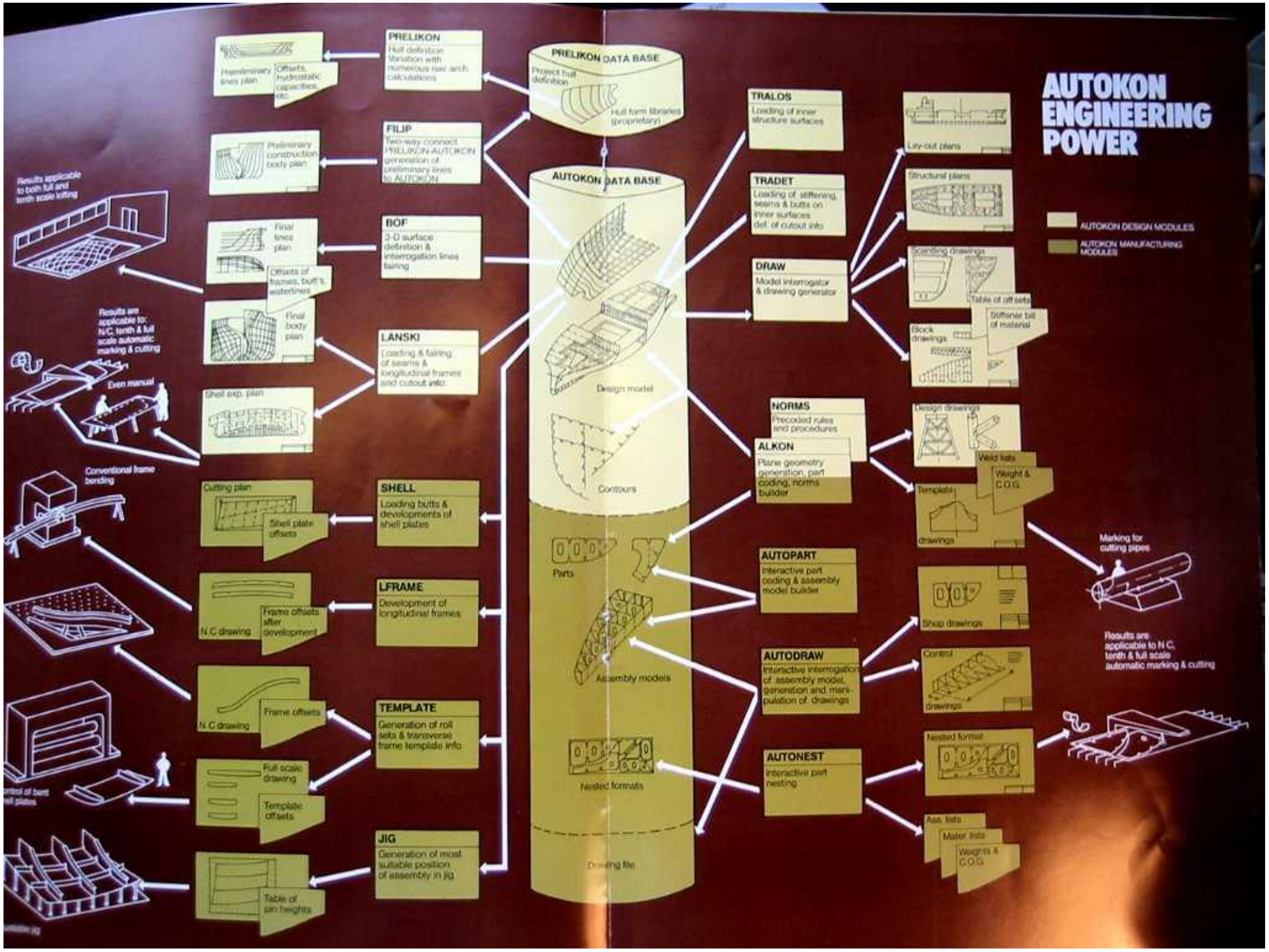
- We must, however, consider some other unresolved questions of great importance:
- It all boils down to a few sets of questions / points:
  - Organisation of the system surroundings (users etc.)
  - Stability of the organisation
  - Stability of functionality
  - Economy (initial and over time)

.The answer to these points will normally put up limiting boarder lines for the system and thus for the models.

Example - Reliability



Example: ”apparent” common DB


AUTOKON ENGINEERING POWER
AUTOKON DESIGN MODULES
AUTOKON MANUFACTURING MODULES
PRELIKON DATA BASE
Project hull definition
Hull form libraries (proprietary)
AUTOKON DATA BASE
Design model
Contours
Parts
Assembly models
Nested formats
Drawing file
PRELIKON
Hull definition Variation with numerous nav. arch. calculations
Preliminary lines plan
Offsets, hydrostatic capacities, etc.
FILIP
Two-way connect PRELIKON-AUTOKON generation of preliminary lines to AUTOKON
Preliminary construction body plan
BOF
3-D surface definition & interrogation lines fairing
Final lines plan
Offsets of frames, butt's, waterlines
Final body plan
LANSKI
Loading & fairing of seams & longitudinal frames and cutout info.
Shell exp. plan
Results applicable to both full and tenth scale lofting
Results are applicable to: N/C, tenth & full scale automatic marking & cutting
Even manual
Conventional frame bending
SHELL
Loading butts & developments of shell plates
Cutting plan
Shell plate offsets
LFRAME
Development of longitudinal frames
N.C drawing
Frame offsets after development
TEMPLATE
Generation of roll sets & transverse frame template info
N.C drawing
Frame offsets
Full scale drawing
Template offsets
Control of bent shell plates
JIG
Generation of most suitable position of assembly in jig
Table of pin heights
TRALOS
Loading of inner structure surfaces
TRADET
Loading of stiffening, seams & butts on inner surfaces def. of cutout info
DRAW
Model interrogator & drawing generator
Lay-out plans
Structural plans
Scantling drawings
Table of off sets
Stiffener bill of material
Block drawings
NORMS
Precoded rules and procedures
ALKON
Plane geometry generation, part coding, norms builder
Design drawings
Weld lists
Weight & C.O.G.
Template drawings
Marking for cutting pipes
AUTOPART
Interactive part coding & assembly model builder
Shop drawings
AUTODRAW
Interactive interrogation of assembly model, generation and manipulation of drawings
Control drawings
Results are applicable to N.C, tenth & full scale automatic marking & cutting
AUTONEST
Interactive part nesting
Nested format
Ass. lists
Mater. lists
Weights & C.O.G.

Another system model

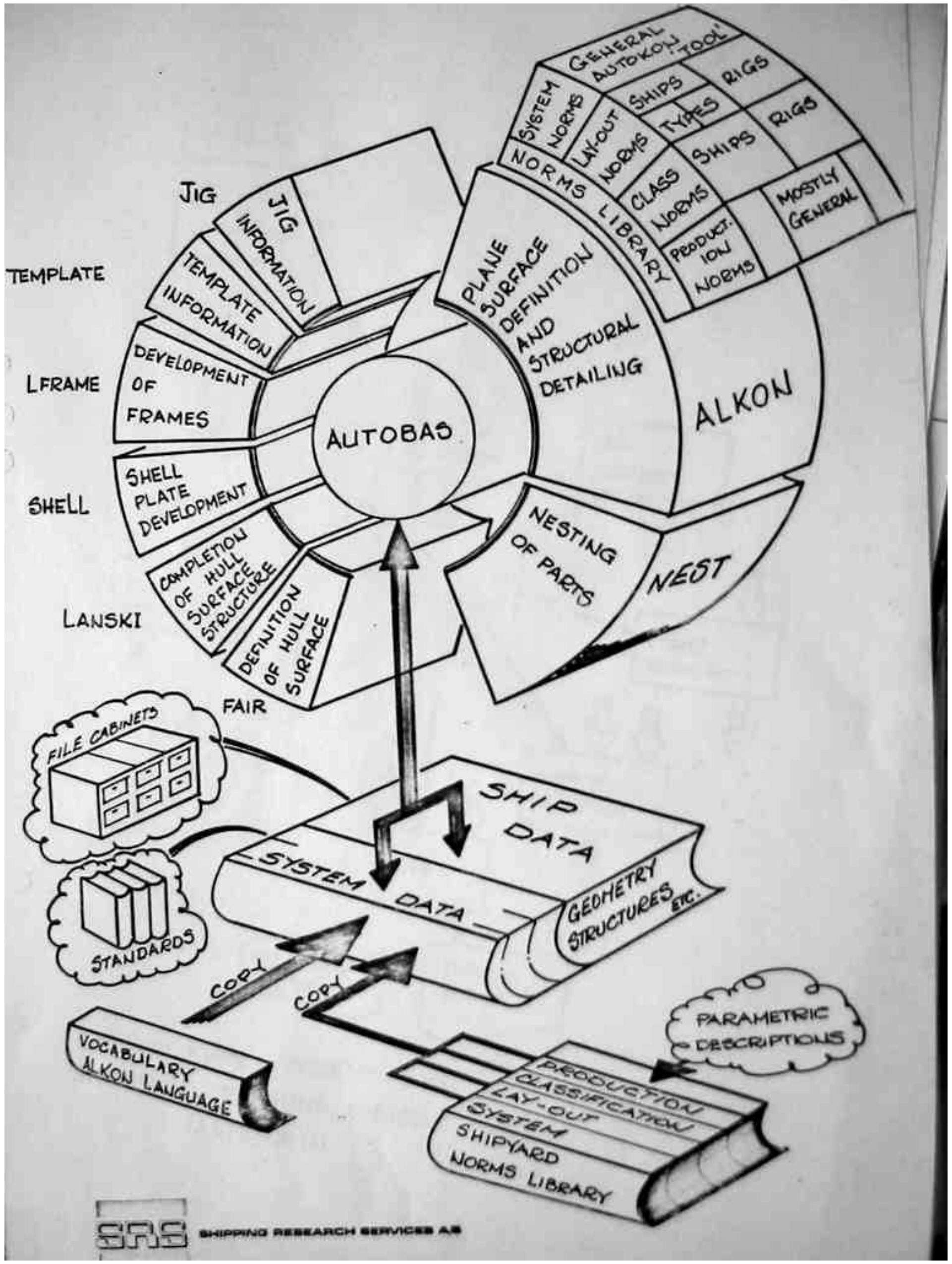
The AUTOBAS contains two major types of data:

Permanent

Temporary.

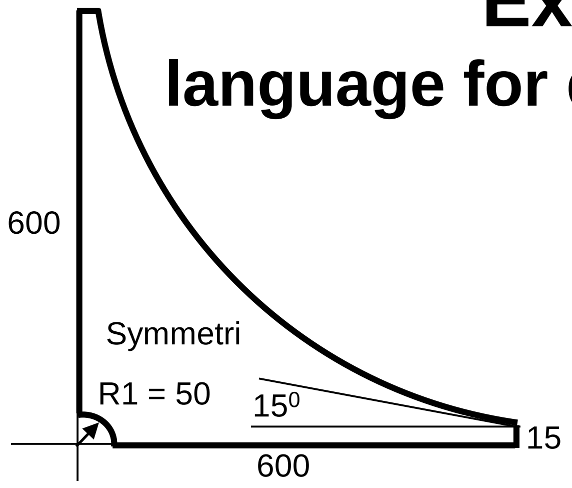
The *Permanent* part works very much like a traditional DB.

The *Temporary* part is a work area and an overflow area (e.g. for arrays), holding structures with both FIFO and LIFO behaviour, stacks and heaps.



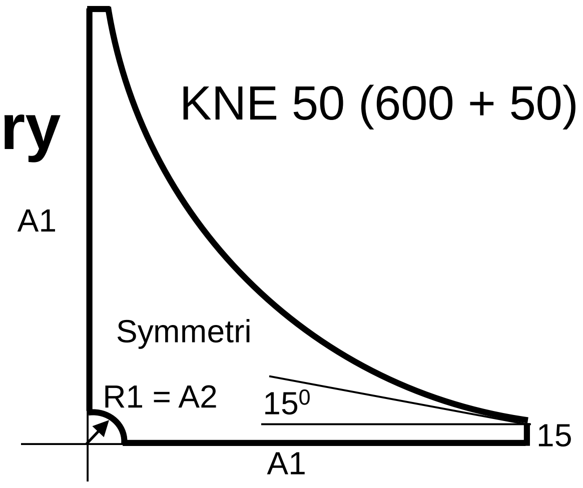
Anton H. Landmark

Example:
language for defining Geometry

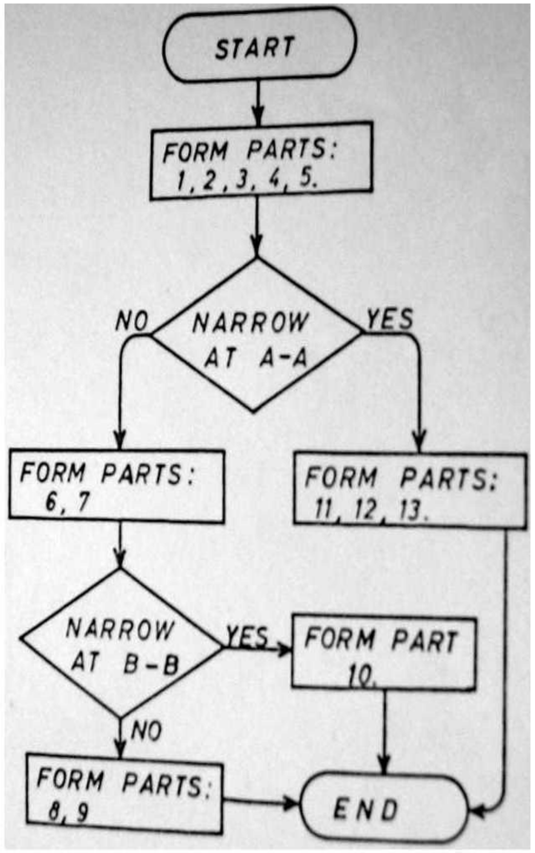


```
START LCON  (+50 +0)
SL: LGTH (+600) DIR (0)
SL: DIR (+90) LGTH (+15)
CIR: SDIR (+165) EDIR (+105)
   EPT (+15+650)
SL:  EPT (+0+650)
SL: EPT (0+50)
CIR: CNT(+0+0) ROT(-1) EPT (+50+0)
END LCON>
```

KNE 50 (600 + 50)

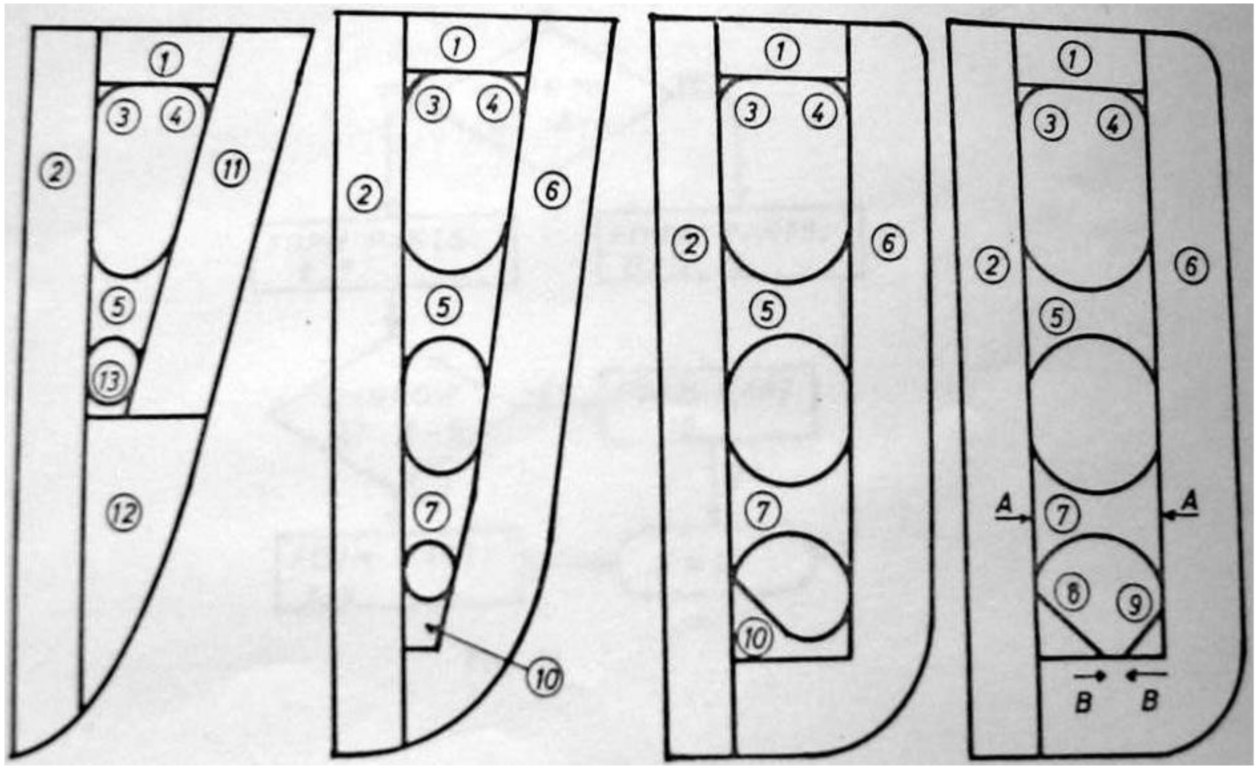


```
START  NORM >
START LCON  (+A2 +0)
A6 = (A1+A2)
SL: LGTH (+A1) DIR (0)
SL: DIR (+90) LGTH (+15)
CIR: SDIR (+165) EDIR (+105)
   EPT (+15 +A6)
SL:  DIR (+180) EPT (+0+A6)
SL: EPT (0+A2)
CIR: CNT(+0+0) ROT(-1) EPT(
+A2+0)
END LCON>
END NORM (KNE 50)
```



START
FORM PARTS: 1, 2, 3, 4, 5.
NARROW AT A-A
NO
YES
FORM PARTS: 6, 7
FORM PARTS: 11, 12, 13.
NARROW AT B-B
YES
FORM PART 10.
NO
FORM PARTS: 8, 9
END

Web NORM



1
2
3
4
5
11
12
13
1
2
3
4
5
6
7
10
1
2
3
4
5
6
7
10
1
2
3
4
5
6
7
8
9
A
A
B
B

Prosessor stacks

§ COMM (a manuscript)
FRAME 16 ((TVfrm 12) +(TVspt 26) +0,5)
FRAME 16 ((TVspt 27) +(TVspt 36) +1)
BOTTOM 24 ((TVspt 14,33) +(TVspt 24,33) +1)
BOTTOM 24 ((TVspt 14,67) +(TVspt 24,67) +1)
&

§ COMM (def. of FRAME 16)
START NORM ’
NEW SBUF’
START RCON SBUF
(***contour code***)
SPT (+10+20) SL:
-
NORM 218 (+ + +)
-
END RCON’
STORE SBUF (***name***)
ENDNORM (FRAME 16)
&

§ COMM (def av NORM 218)
START NORM ’
NEW SBUF’
START LCON SBUF
(***matrise***)
SPT (+10+20) SL:
-
KNE 400 (+ + +)
-
END LCON’
STORE SBUF (***navn***)
ENDNORM (NORM 218)
&