BIT/RC: Back to IT-basics / Refresher Course

BIT/RC is dedicated to my son Pieter.

Dear audience,

"BIT/RC: Back to IT-basics / Refresher Course" is primarily intended for those wishing to refresh their knowledge about Information and Communication Technologies (ICT).

For example: if you are a post-graduate student who wants to revisit what you learned in school then this course will give you a very solid foundation. However, any other person, including the non-graduate, will be able to determine if it is worth their while to look around the ever-evolving world of computer science. In addition, it would be appropriate for a manager of computer developers and other technicians to undertake this course to better understand his staff without feeling completely lost when surfing the internet's wiki-pages.

Of course, if you really want to code in JAVA, C++, C#, COBOL etc. then you must dig deeper into the dark side of "bits and bytes", where only computer-nerds seem to have fun. If this is your intention then you might appreciate the attached description of the ABSOUV-project, and if this still grabs your attention then feel free to ask me for the actual code that I wrote for a prototype version. Otherwise, please ignore that part of BIT/RC completely.

Sincerely,

Willy van Remortel

Main parts of the refresher course.

All text in this document is based on the architectural insight of its author, although limited to administrative applications for multinational companies. BIT/RC is dividied into the following 8 chapters:

- Batch Processing (operator console).
- Files and jobs (scheduler software).
- Databases (data definition language).
- On-line Processing (transaction commitment).
- Libraries (interface software).
- Connections with external systems (network lines).
- Programming Languages (development environment).
- Standard Solutions (enterprise package).

Appendix: the ABSOUV-project description.

Contact via: willy.van.remortel@gmail.com

Feedback via: mikemadden1@gmail.com

Foreword: jumping from one thought to another (my hobby).

For centuries human beings have been making tools, machines, computers, robots: no other animal has done achieved all of this to date. Although we cannot fly like birds, we get to the moon and they do not. However these characteristics are not present within all of us, and therefore they cannot be seen as another milestone in the evolution of our life form. Are other products of our brain, such as fantasy, dreams, language, works of art, the correct attributes to definitively distinguish us from animal precursors according to the theory of evolution?

Without formulating an answer to this question, I will jump to another thought: do humans and apes start with the same package of genetic material? After birth they resemble each other physically for a considerable time if they are from the same evolutionary branch, but on the spiritual side their differences soon become apparent. Will tangible characteristics of chemical processes ever prove this in a laboratory-test: the exploration of the physical function of the brain keeps us busy already for a long time and spiritual functions will maybe never be disclosed.

Why does the evolution and chance of survival stop for elephants, dolphins and great apes, for example, except for environmental factors? Viewed in this way, "evolution theory" would be better defined as "limitation theory" if species are compared to the human mind.

Is our genetic material hidden for a long time in the determining the variant that will transcend our limits in such a way that it will later serve as the next step in our evolution? What happens to the descendants of humans who never achieve the minimum performance? For other life forms, we gladly and naturally take a step back in time to detect the moments of their first appearence. In this area, scientific evidence and religious dogmas actually reach the same frontier of our thinking ability, to get through to the origins of life, let alone to impose a vision of such a future fact on unbelievers.

Let's jump to another thought again now: can we learn something from the creation of computers? They are very similar to each other in terms of internal workings expressed in "bits and bytes" but they differ greatly in terms of applications expressed in software. In other words, the fundamental difference between, for example, an ape and a man is not restricted to the physical functioning of their brain, it extends to the psychological insight into their behavior. Is the intelligence limit already established for all people, ever since their first genetic specimens walked around on our planet? They were not all geniuses, I would speculate.

A final thought: why should there not be such a thing as the "table of human behavior" by analogy with the "table of chemical elements (Mendelejev)"? For example, can a computer application help us to collect biographical material to extract the essential behavioral characteristics from it? Does the inevitable new species of humanity show a higher ethical behavior that far exceeds the current level? Hopefully, the warlike variant will die out. Of course, I dare to think that such an event is not fullfilled via the "limitation theory": this would mean humans master the elimination of bad people on our planet.

Table of Contents

1. Introduction to BIT/RC.

1.1. Purpose of BIT/RC.

The aim of the course is to refresh the basic principles of "Information and Communication Technologies (ICT)". After completing the course, participants will be able to answer technical questions at school education level.

- Americans do not use the C for Communication and therefore refer to IT rather than ICT, whilst the French prefer the term "l'informatique". However, the C is a very important component as you will see throughout the course.

1.2. Constraint of BIT/RC.

BIT/RC is limited to the automation of administrative business processes. For example:

- The application field that deals with the automation of industrial production processes or robotisation is not mentioned anywhere by BIT/RC.
    - Nevertheless, at least one external connection (network) with an administrative service is guaranteed to be in use by almost every factory. It could be so for:
        - forwarding customer and order data to production machines.
        - receiving results from a laboratory.
        - the centralization of data in general, if necessary with the aim of synchronizing systems.
- BIT/RC contains a chapter on "external links": they are the point where data enters or leaves the computer via a communication line that is not always fully controlled by the owner. Obviously, security mechanics, like a firewall, prevents the hacking of a computer. Hence, a communication line with several layers upon it is managed quite differently from an administrative business process.
- Computer applications for hobbyists, for example games, or for home use (domotics / internet of things - IoT) are not referred to explicitly within BIT/RC.
    - This is not really an issue because BIT/RC gives a complete basic knowledge of large and medium-sized systems so that you can switch easily to gathering knowledge about a home network with, for example, hand-held devices (smartphone), a printer or disk-storage.

1.3. Approach by BIT/RC.

BIT/RC is effectively a self-teaching system. This needs to be understood as, for example,

- (1) your questions are not answered in a dynamic way by a teacher in a classroom.
- 2) each chapter is structured in such a way that you can ascertain an answer to the question "what is it for?" followed by "how does it work in general?" and "are there specific examples?".
    - After reading the examples the user can enhance their understanding with detailed information via the internet (relevant search terms are indicated in blue).
- (3) ideally, you will be in a target group that by default comes into contact with very old centralized systems (for example: banks, insurance companies, multinationals that began their automation business a long time ago).
    - Only when the ancient developments are well understood is there room for exploring modern techniques and their impact on applications i.e. old business rules and new requirements must always remain coherent.
    - Conversely, departing from contemporary jargon and the very latest hype in architectural design, will cause much significant confusion as many previous decisions were made at a time when computers were still far too expensive and their management was very impractical. For example:
        - Without network capabilities there were only specicalists in data entry via punch-cards and processing via very limited programs, oftenly "one at a time" in just one system and with business logic spread over many small steps. The gap with upper management was huge in the past.

1.4. Experience of the author of BIT/RC.

My view on computer science is based on more than 45 years of experience in this field. My background incorporates the following occupations (which includes 20 years in the permanent service of the French computer manufacturer Honeywell Bull, more than 10 years at the Belgian branch of Cap Gemini and Sogeti and the remainder with freelance assignments in multinational companies in Luxembourg, The Netherlands, Great Britain, America and Canada):

- operator and programmer for mechanal systems (hardware-relay), functional analyst for administrative systems (general ledger / social law / logistics), project manager for outsourcing (small and medium-sized enterprises), business analyst for very large companies (bank / insurance), software distribution manager, trouble-shooter for huge mainframes (dump analysis / hardware failure detection), teacher in an international

school for software engineers and developers (Assembler / COBOL / SQL), consultant for relational databases (early adopter), SAP consultant for corporate data management (R/2), support engineer for UNIX systems (before the Linux flavour), application manager at the Telecommunications Centre of the European Commission (C-language), and finally ICT-architect for the automated migration of platforms (downsizing / distributed systems / translation rules for computer languages).

1.5. About the practical part of BIT/RC.

The practical part of BIT/RC has a very ambitious aim: implementing file-content manipulation at byte-level by end-users who do not need to understand any of the current or legacy programming languages.

I started the ABSOUV-project in 2015 with the idea of constructing a software package that oversees the ICT-field from multiple positions or specializations.

- Moreover, in this refresher course I approach all parts from a top level, where it remains understandable for everyone, all the way down to the minute details, where only the "nerds" are really happy.

In the appendix I describe the idea for a product that would potentially make programmers superfluous.

- However, here I want to stress the need to find computer specialists to build the final version (intended for the consumer market).
    - The ABSOUV-project will be in line with the current climate of the most important trend since the creation of the internet, namely "cloud computing", which is nothing more than combining the very old concept of centralized hardware and software together with very modern methods of communication.
        - The conversion of user requirements to an application runs cyclically with possible corrections or modifications in the software: the language that is part of the ABSOUV-project offers end users the possibility of completing a cycle without the help of analysts/programmers. So, there is no need to first record everything in a formal and rigorously structured environment suitable for data-entry, storage and output processing.

Furthermore, the practical part of BIT/RC shows "step-by-step" how software engineers would create an extensive product based on just a few definitions: at the end this would require very little technical knowledge from end-users who otherwise must learn how to program.

- The ABSOUV-package aims to avoid 2 essential problems that occur with almost every IT solution.

- Firstly, the usual design techniques are based on the limits of the now obsolete architecture (infrastructure for disk storage, memory, connectivity, user interface, etc.) As a result, the reformatting of data entry continues stubbornly onwards at the expense of a clear representation of the original data within the associated documents.

- Secondly, a knowledge gap always appears between the (usually simpler) wishes of end users, against the deployment capability of highly specialized analysts and programmers: they essentially convert the user's business logic into a complex computer language (COBOL, JAVA, C++, etc.)

    - Hence, the ABSOUV-package introduces a new strategy that brings together, as closely as possible, a theme created by an end user and the underlying computer program.

    - In fact, the lines of code are generated automatically according to the specifications in ABSOUV's scenario text. The end user, who is unable to read or write computer programs, does not care if the generated code language is JAVA, C++, C# or COBOL, nor does it matter to them if data-storage deals with integers, alphabetic characters or floating-point numbers.

2. Basis "Information and Communication Technologies (ICT)".

There are several ways to explain business applications on a computer: the explanation to beginners is always very different from that given to senior management, and end users are usually given a very specific approach (note that I refuse to use the term "for dummies").

- Hardware and software are evolving together towards major improvements in the area of infrastructure (platform). Naturally, new applications can benefit from this evolution immediately, but they usually only survive thanks to the help of very old concepts (commodity hardware and software / legacy code). These concepts are divided into 2 groups: batch and on-line.
    - For that reason, it is very important that you properly understand when an application is suitable for batch processing and when on-line processing is appropriate. The role of external connections, worldwide, must also be part of your basic knowledge.
        - ICT will be used in the future to develop non-administrative applications, including artificial intelligence (AI). It might be given a different name, for example: internet of things (IoT). However, behind all modern technical solutions the old concepts serve as an underlying foundation.
    - Another very important concept to understand is the meaning of "architecture": in many cases people associate this word to the way solutions are built, for instance, "service oriented" or "framework".
        - Solutions can be ported to another hardware/software platform by the translation of the coding lines and by the migration of the data-storage.
        - In such cases the basic elements remain the same, in other words, a foundation for different types of solutions.
        - This is why BIT/RC will present to you the foundation of a computer architecture and not to one of the many flavours of solutions constructed on the basic elements.

2.1. History: the computer without communication.

The history of a helping machine for the automation of administrative procedures, for instance "processing data by computer", shows only one major change, and that is brought about by

the addition of a means of communication: all other technological innovations increase the performance and efficiency of the information processing, but they do not offer end users a totally new concept about the underlying "bits and bytes". So, this new concept of computing was revealed to the public in the 1970s, and no other major element has been added to the foundation of computer architecture since (although, of course, there have been many new solutions built on top of the basic elements).

- The old concept, that predates 1970, is based on four basic elements: (1) a central processing unit better known as "processor" (2) the equipment that supplies input and output, mainly known as peripherals (devices) (3) the logic or software, and finally (4) the required staff.
    - With these four elements, a company can automate administrative procedures and manage all kinds of information. This concept translates into the range of computer manufacturers that work with a model series, and that should support the growth of information. For example:
    - There is the choice of either a large (mainframe) or a medium (midrange) or a small (personal) computer depending on the amount of data to be processed (volume).
- The new concept adds the element of communication, with all manner of end users connected, as the 5th basic element.
    - This gives virtually everyone access to information instantly, i.e. as if it were an "on-line" conversation with the program that runs within the computer and has just processed the data. That translates into a huge spread of infrastructure (multiple platforms). For example:
        - All kinds of machine models are present simultaneously in a solution that is at the service of all company employees anywhere in the world.
        - Not only is data volume important, but also the flexibility with regard to the number of connected end users.
        - This leads to many new ways of data processing, which increases the workload for an infrastructure architect as well for an enterprise architect or a solutions architect.

2.1.1. Basic element: processor.

In 1950 a computer was mechanically driven by a motor with gears and relay circuits: this basic element acts as a processor because it allows commands to be executed in a completely automated way, and it enables repetitive process, also known as cycles. For example:

- Within each cycle, a number of relays are activated as the result of a card, punched with holes, making contact with 80 brushes, storing a representation of up to a maximum of 12 elements (rows 0 to 9 plus 11-th and 12-th overpunch).
- The circuits are further controlled by wiring via an external programming board. At every major intermediate step in the logical processing (the transfer of the results of calculations) a new board must be mounted on the machine.
- In the 1960s, the motor was replaced by an electronic processor and relays were replaced by transistors. The operating principle also became more extensive and more powerful. For example:
    - The 150 cycles per minute of a mechanical motor evolve to vibrations of a crystal, processor and today the computing power is expressed in the number of MIPS (million instructions per second).
- The logical principle established in the early days of information processing by machines is retained still retained today. For example:
    - A relay that passes electricity has an open (OFF) and a closed state (ON) which is simulated by a transistor that passes direct current: as the hardware becomes smaller (via miniaturization), many electronic components are put together in one integrated circuit (IC / chip).
    - Even today the logical operation of computers is still based on OFF and ON (zeros and ones of the "binary system"). For example:
        - The on-off principle provides the basis for an accumulation of logical decisions and calculations in one machine instruction (mathematics / boolean algebra).

The concept within the processor that replaces the thought processes of humans with regard to gathering and producing information (business logic) is achieved by the following basic elements:

- Algorithms written in a logical language that is translated into machine instructions (software).
- Enabling high volumes for the input and output of company data (peripherals).
- Management of data from a central system (computing center personnel).

Consequently, a company purchases either a very large (mainframe), or a medium (midrange) or a small (personal computer) machine, but always aligned with the same principle of the 4 basic elements focused on data volume.

2.1.2. Basic element: peripherals.

With the emergence of a large internal memory for the storage of results from the processor, the automated execution of several programs could occur in sequence, without a physical change of wired-boards: at the same time, magnetic devices were added to the machine that stored the results more efficiently and made them much easier to reuse.

- Initially, memory consisted of ferrite cores for internal access up to a maximum of 1024 bytes (RAM) and it was not until much later that memory banks of multiple giga bytes were created. What a difference to the "80 times 12" punch-holes!
    - A punch card was the first way of inputting data (80 characters), then a typewriter (TTY) was connected to the computer, and eventually the TTY was replaced by a screen and keyboard (console).
    - The logic around "bits and bytes" developed in the old concept remains the same all the time. For example:
        - A hole in a punch card corresponds to a number, a combination of 2 holes in the same row yields a character, 3 holes denote special signs such as a point or comma. The digits and letters have been replaced in modern equipment by a fixed configuration of bits derived from the original combinations in punch cards (character set).
    - The first output peripheral computer connections were are used for punching holes in a card so that they could be mechanically sorted with another wired machine or mixed with "mother cards": data that contains an identification number for customers, products, suppliers or any other stable object within a company (sort / merge handling).
        - The next development was magnetic tape that replaced punch cards, and this was followed by the introduction of the hard disk. These were used, for example, for a "sort / merge" software process delivered by the computer vendor.

Hence, peripheral equipment (I/O-devices) serve as input and output of the memory processed by a program: they are located near to the central unit of a computer via cables which have a very limited length (I/O-channels). Apart from the amount of memory, computers do not differ from each other. For example:

- With a large machine (mainframe) a very large number of "channels" are available.
- With a small machine there are only a few "channels": one for the connection of a hard disk, optionally one for a magnetic tape and so on.

2.1.3. Basic element: program.

In the early days of the development of business software the product was inevitable complex because the programming language was very close to machine instructions (ASSEMBLER). A few decades later, more readable languages, closer in nature to written and spoken English, appeared (COBOL and FORTRAN), but they still required a great deal of expertise in the field of logical thinking, including the ability to build up abstract reasoning about the observed business processes.

- Programming is the cutting of a problem (Specifications of a solution) into ever smaller parts, and then bringing the whole thing back into the automation of an information flow within a company (business process flow).
- Today, the profession of computer scientist is still based on a person's analytical and synthetic thinking ability. There is no such entity as "an easy to learn programming language". However, today's programmers do not need to re-write standard functions every time a project starts. For example:
    - Due to the need for simplification, the C-language has arisen, in which a lot of logic is stored in common libraries. This technique saves a lot of time and effort: the C-programmer simply needs to master the interface into the libraries' functions.
    - Building on this technique, object-oriented programming with C++ allows a much better reuse of components. More recently, cross-platform development has been added and portable code (Java is such a language).
        - Each new element of invention results in a far-reaching specialization for programmers, so that a comparison between today's technology with the degree of complexity for assembler is ultimately irrelevant: the training for a programmer apparently still requires a very good understanding of a computer's architecture.

2.1.4. Basic element: staff.

In the early days, all of the tasks related to computers was outsourced by the customer to the manufacturer of the computer because of the scarcity of trained personnel. Consider the following phases of a project at that time:

- Interview with a subject matter expert at the customer's site. This usually involved an accountant, the smartest person in the department or a senior manager: none of them needed to have any understanding of ICT, but they brought their understanding of the company business to the project.
- Validation of the data inventory for the specified functionalities in general, and the very specific requirements of end users invovled in the business-flow (user requirements).

- Scoping of a complete solution (usually split into sub systems because the capacity of the computer was too limited to process everything in a single cycle or " RUN ").
- Design of software with pseudocode and / or diagrams.
- Coding and testing.
- Delivery of the computer with all of the customized software, and training of the end-users.

Even the control of the computer (operator's console) and the use of the applications (production of punched cards as data entry) were often outsourced by the customer to the hardware vendor. For example:

- In the 1950s the changing of programming boards was almost a full-time job along with the repeated re-sorting of large trays of punched cards in order to get to subsequent steps and ultimately the final results.
    - The printed results are the only artefact that is presented to the end users: it demonstrates the transfer of information internally within the company (information flow / manual procedures).

2.2. Evolution: the computer with communication.

Within these old concepts, for almost every company, the computer's range was limited by the restrictive length of just a few metres of the feed channels for peripherals to the central system: in 1970, the virtually universal abolition of such restrictions led to a new method for the input and output of data. This meant a significant evolution in IT, symbolized by adding the middle C: from that point on, a communication line was the fifth basic element in every computer-system worldwide.

- It was also the start of an architectural approach, explained in the next chapter, instead of a purely infrastructural view of basic elements in either a mainframe, midrange or small computer.
    - The addition of the new basic element resulted in a new concept for data processing. This meant that several computers could participate in a solution for the automation of administrative procedures.
    - No other change happened before or since that had such a significant impact on the computer (please, let me know when you spot the six-th element).
- The first significant influence in this respect, something that is evident to everyone, was the emergence of peripheral connections for input and output, intended for employees at the same geographical location, within a radius of hundreds of meters.
    - This initially leads to the emergence of an intranet and eventually, around 10 years later, the internet: workplaces are now spread across the entire planet

(hence the world-wide-web) connected via physical communication lines or wireless (Wifi).

- At the beginning of this new era in computer science, a computer (a) consisted of a central system i.e. a mainframe or a mid-range computer according to the required computing power and volumes, and (b) of wired endpoints (TERMINAL-devices) where all data entered (via keyboard) can be shown back to the user (via display / screen).
- Other more recent innovations include adapted machine interfaces in the form of a mouse (via pointing device).
    - Many small devices (handheld devices), such as a smartphone, are currently used for the creation of data, just like an old-fashioned "terminal" did in the 1970s.
- A new industry for user-friendly infrastructure (middleware) was created that could even convert spoken words into text files, or recognize handwritten text (scanner / OCR).
- The most famous display from the early days of communication is a “3270” TERMINAL. This was called a "dumb terminal" because the strategy to obtain a flow of data was very simple. For example:
    - A screen receives a prompt from the central computer (eg a mainframe): there is a square mark on the screen that points to the starting position for sending back and forth characters (cursor).
        - This allows typed characters from the keyboard to appear on the screen after reception by the mainframe, which sends the typed characters to the display (the echo property of the asynchronous communication protocol).
    - A much faster facility saves a stream of characters (buffering/synchronous protocol) until the special ENTER key is pressed.
    - After pushing the ENTER key the mainframe uses an application (ie the "business logic" written by the company's programmers): lines and columns are sent to the TERMINAL (form to be filled out via the user's keyboard).
    - The user (terminal operator) completes the FORM on his screen and the application checks the input, writes output to disk, and starts a new cycle for the same FORM or for another one
- When analyzing a server today, it is clear that it uses an identical strategy for starting a communication: there is a program that waits for input to continue a conversation with an intelligent computer instead of a dumb terminal.

The back-and-forth communication (conversation) between mainframe and terminal therefore proceeded according to a master-slave protocol of which the main features and basic techniques still remain valid today.

- As the number of connections increased exponentially, the manufacturers of computers designed a special package (transaction monitor) to allow a session with multiple FORMS for each terminal user without impacting other users.
    - A session acts as if only one terminal (end-user) is engaged with the mainframe and thus keeps the data or interim results unique for that user intact from the point that the user logs in (login) until the moment they log out.
    - The protection and integrity of data within a session is also applied in database techniques.
    - There were therefore solutions for simultaneously updating data in a database with COMMIT and ROLLBACK commands for controlling the integrity for applications that run simultaneously.

Finally, the communication technique has evolved into a worldwide network using the client-server method. Seen from the architecture's point of view, this facility offers a reliable solution for keeping a database up-to-date. For example:

- The SERVER is a midrange computer (UNIX system / service port) that waits for incoming CLIENTS using the socket protocol (a logical connection between applications that remain coupled for the duration of a session).
- The CLIENT is a Personal Computer (PC) that contacts the server's port in order to start a session.
- The internet (with a BROWSER program as CLIENT) is a variant on client-server for which the http-protocol does not require any facilities to manage a session for each user. Nevertheless, solutions have also been found in the form of objects that are temporarily stored somewhere between the SEND and RECEIVE of data (storing a context during a commercial transaction).
    - In line with the possibilities offered by the internet, a company can opt for the use of storage infrastructure without purchasing or managing it itself (via the CLOUD).
- The use of devices which are not directly connected with a central system when data is collected leads to new methods, meaning a new architecture for a company's solution. For example: synchronization via the transfer of a flat file to the company's central system, or the use of a transfer service that does not even belong to the company (web-service).

2.3. Trend: professionalization.

The evolution of communication tools also became evident in certain activities, that led to some employees, without extensive knowledge of ICT or development tools (subject matter experts / SME), can build applications with office software such as a spreadsheet.

As a result, the professionalization in every large company has led to the parallel execution of several projects, whether connected to the mainframe or not, as follows:

- Application and allocation of the company's budget to build or expand a new application. This is often accompanied by a SOW (STATEMENT OF WORK) that is drawn up by internal or external consultation, and this can also evolve into a "proof of concept" (POC).
- Elaboration of a complete solution according to standard / standards of QA (QUALITY ASSURENCE) as applicable to the whole company.
- Outsourcing the testing to an external supplier (black box/white box). There is also the possibility of "outsourcing" the complete operation (PRODUCTION STAGE) according to standards fixed in a Service Level Agreement (SLA).
- Use of a new or renewed application on various hardware and software platforms simultaneously (distributed systems / hybrid solution).

2.4. Complexity: specialization of computer-engineers.

The separation of the profession from one computer engineer into various specializations must primarily be seen from the perspective of infrastructure. For example:

- which peripherals (devices) with their associated operating system belong to the company (purchased or leased)?
    - In a small company, such a question does not arise, because one person has to manage the production within ICT. As a result, specialization is not efficient and external IT specialists can be called in to solve "non-daily" problems.
    - In a large company it is efficient to work with computer engineers who are exclusively involved with PCs, with another team to take care of the daily production of a mainframe. A cell with communication / telephone specialists is often also a necessity to intervene quickly when problems arise.
- In many cases, a steering committee at the highest management level (project sponsor) will determine the division of labor for everything that has to do with new projects within ICT (ALM - application lifecycle management).
- The project manager is usually a SME who does not have to have a broad knowledge of ICT but who must guarantee the feasibility.
    - However, to build an application, a decision must always be made beforehand about the part of the infrastructure that will later facilitate the exploitation of the data.
        - As a result, the influence of IT specialists remains considerable to keep costs and benefits in balance (business case).

- After completing a project, the application is eligible for maintenance (with a SME as the person responsible).
    - Therefore, engineers that only make minor changes are deployed in consultation with the production department that manages the entire infrastructure within the company (ITIL - information technology infrastructure library).

- The collaboration between a specialized engineer and the company's SME should therefore essentially be viewed from the perspective of an application (data owner):
    - ICT is just one of many tools to support a company's core activity in respect of data collection.

2.5. Optimal solution: team.

The knowledge of the activities of a company (subject matter) always play the most important role in any ICT project for the automation of an administrative procedure.

- The composition of a project team (to build something new or to change something significantly) must always be envisaged by viewing the final application.
- Specialized engineers should only be called in according to the need for knowledge about the infrastructure and a feasible solution. For example:
    - is there a need for a small configuration or a very large or decentralized equipment to host the software?

So, a team must be constructed according to the development effort of an application and its use. For example:

- A very large central mainframe (the successor to the large iron skeleton from the early days but now more compact and with thousands of connections to the outside world) needs the following experts.
    - An operating system engineer (administrator).
    - A specialized engineer for the database (DBA database administrator).
    - A specialist engineer for ON-LINE processing / transaction processing (TP administrator).
    - Several developers (programmers) optionally skilled for maintaining the inherited coding from the past (LEGACY).
- A network that serves users locally and also provides remote access with appropriate security (firewall , authenticity check, limited services).

- Administrators of multiple servers and the HUB with network connections and TELECOM (provider).
  - For some time there have been combinations of computers from various manufacturers (each with their " OS operating system " (of which WINDOWS is the most widespread due to home use), and programming languages (of which COBOL is the most common due to its LEGACY in very large companies).
  - Every OS and programming language requires specific training for programmers. JAVA, as an intermediary, tries to break through this problem with a virtual machine on multiple infrastructures (JVM - java virtual machine). Elsewhere, a competitor almost established a monopoly with their own specifications in "CLI" and its derived .Net framework (with C # or c-sharp as the most popular programming language used).

- Developers for ON-LINE transactions (now more common in the form of services offered via the internet in a specific "SOA" - service oriented architecture).
  - Graphical designers (look and feel - standard) work on the presentation side of the application (" GUI - graphical user interface").
  - Other specialists work on underlying services (back-office). These then establish a connection with the central computer for manipulating extensive database data.

2.6. Approach: division of roles.

Professions that were once combined, in the form of one person at the customer's side and one person at the computer-manufacturer's side, became spread over several engineers today so that a project-based approach is always and required.

- The life cycle of a software product (SDLC - Software Development Life Cycle) can seldom be entrusted to the ICT department alone. For example:
  - A project team usually includes subject matter experts (key users) who take charge of a project that involves several departments of the company.
    - These people work together with engineers to automate part of their "business".

- Within the development cycle, specializations arose due to the success of various methodologies or products. For example:
  - iterative / waterfall SDLC, prototyping, agile project management (SCRUM), testing beta versions.

- For ready-to-use software packages, computer scientists no longer need to write programs. For example:
    - End users (under the guidance of a key user) learn to provide the various parts of an "ERP - enterprise resource planning" tool with parameters that are specific to the business of the company.
- Within the profession of programmer there are strictly defined areas for which there is a separate certification. For example:
    - A role such as ".NET developer / C #" is in fact reserved for an employee who only knows the Internet architecture of Microsoft and more specifically the capabilities of the .NET platform. By passing examinations such a person gets a quality accreditation.
- Globalization demands a new way of cooperation. For example:
    - The business analyst with knowledge of computer science, or in some cases a subject matter expert, with the help of a functional analyst, can create all of the specifications of the application for the ICT developers who can be based in another (often cheaper) country.
    - Testing of finalized software (black box testing) is left to non-IT specialists, which means that the cost price is reduced.
        - With extensive experience, even this part of an SDLC can be partly automated.

3. ICT Architecture.

3.1. BIT/RC's definition.

ICT architecture in general is a way to clearly map complex infrastructures of collaborative computer-systems.

- An architectural method offers opportunities to deal more efficiently with "human resources" and the sharing of tasks within multidisciplinary teams (project management).
- The purchase of infrastructure (hardware and software) has to fit in with the vision of the management with regard to "doing business" with the outside world (supplier market / customer market) and more and more efficiency is demanded, therefore, in large companies the role of an ICT-architect varies from technical-oriented to solution-oriented activities.

BIT/RC is limited to the observation of administrative processing. This means is specifically the automation of information flows within a company.

- An information flow starts at a given business department and usually ends at another one.
    - To collect and pass on the data, people use documents to which rules are applied for completion. The processing of such information (business logic) provides new data and possibly additional documents.
    - Such manual procedures lend themselves perfectly to automation, where the business rules are cast in computer programs, meaning: they are hidden in an abstract language, incomprehensible for end-users dealing with the manual work.
- In the context of BIT/RC, no subjects are discussed regarding the import or export of data in specialized production facilities on the industrial shop floor (factory / plant).
    - Nevertheless, the ICT architecture will have a major influence after an external connection has been established (usually via a local network) with robot systems, for example:
        - the specifications of the formatting and the volume of data will require investment in additional infrastructure (eg devices, software, data conversion).
- The alternation between production processes (conveyor belt, refinery, logistics, packaging) and administrative processes can only be understood by all concerned if there is an overview of all used hardware and software, especially with regard to their cost price in both purchasing and management (including program maintenance / data production).

3.2. How to examine the needs for automation.

The approach to a company´s automatization by way of an architectural description, always starts at the top (helicopter view) with thre answer to one simple question:

- where do you buy or lease something that involves ICT-people?

The answer is of course simple: "at a seller or vendor" (followed by the names of suppliers), but then the counter-question immediately pops up: "what should it serve?". Only then will there be a logical answer that will allow a further breakdown for ICT: the best answer is that it must serve to build all applications that meet requirements from various departments within a company.

- When user requirements are too high, senior management must decide on the purchase of new ICT resources.
    - The needs of a company are most visible during a thorough study of the information flows (business process), regardless of whether they are supported by automation. Only after a "business analyst" delivers his document, can pieces of the information flow be automated to seamlessly align with other business processes.
- Sometimes a company grows so fast that it does not have time to think carefully about the appropriate architecture, and "islands" arise: areas where ICT automated information flows that do not fit anywhere in the company's network, and so electronic data cannot be coupled or shared with other users.
- In mergers of companies, exactly the same problems arise as with "islands".
    - A conversion or migration to a uniform ICT architecture within the company then appears to be the only good option in most cases.
        - The choice between "rewriting everything" or automated conversion / migration is rather a matter of money and available tools.

3.3. Fixed framework and logical areas.

BIT/RC's view on ICT architecture always starts with a chart of the infrastructure that applications must use: remember, five basic elements serve as a foundation for any administrative computer-system.

- (1) An architectural view must always consist of a drawing of one or more connected systems, optionally with names of data owners and/or project teams:

- the purchased devices / OS should be in colored blue, the acquired tools / software packages in red, and applications / production jobs / parameters all developed by the company or its consultants in green.

- (2) Always keep in mind that the drawing represents a means to go from a top level to a detailed ICT architecture anywhere i.e. it is a fixed framework for any system all over the world.

Consequently, all major architectures have the same basic principles, regardless of the type of hardware and software that they detail.

- This abstraction of ICT infrastructure is therefore the most suitable place to start with any study or project within a company.

Important note: a sixth element, sensor, for instance for a biometric device, must be added to the fixed framework, especially when dealing with robots or artificial intelligence systems (AI): it is out of scope for this exercise but might be added in a future version of BIT/RC.

Within the fixed framework lies a further division of eight areas that focus on the logical aspects (solutions): in other words, the framework and logical areas together serve as a foundation on which any solution anywhere in the world has been built via today's technologies.

- These areas are the non-tangible part of ICT namely: knowledge and experience (expertise). This non-tangible part of an ICT-architecture comprises 2 main categories as follows:
  - (1) the knowledge about tools with which the applications are built (development environment).
  - (2) the knowledge about tools that keep the applications running (production environment).
- It goes without saying that the eight logical areas are connected to each other and that the detailed view becomes a real tangle of expertise.
  - As a result, there are by definition, not eight ICT experts, but one or more people who specialize in one or more areas.
  - A large multinational company can employ over 1000 IT specialists with the vast majority assigned to change-management (application maintenance).
- The development of an application can be divided into sub-projects according to the ultimate location of processing in the production environment for local data. The

division can also be done because of functionalities that are seemingly disconnected. For example:

  - data flows through from one application to another in phases. Usually a CRUD matrix provides a better insight into the application. A CRUD matrix describes which file or database is written, read, updated, deleted (Create-Read-Update-Delete) and by which components (program units)?

- In general, a thorough use of BIT/RC's fixed framework and logical areas allows a company to obtain detailed estimates of time spent on projects (workload breakdown / top-down approach).

Finally, an infrastructure must be considered in terms of clusters of devices / OS and thus the corresponding tools / software packages and developed applications / production jobs.

- The ICT expertise for one of the eight areas usually serves dozens of components of a multinational, spread all over the world. The view at the top level remains the only good starting point to zoom in on the external connections via a network or the internet.

  - BIT/RC's view about architecture will always zoom in from one central computer to satellites (distributed computing) and the inevitable PC park in a local network area (LAN / corporate office).

  - Next BIT/RC will address other external connection; a wide area network (B2B / file transfer).

3.4. Top level view on ICT architecture.

The division into eight logical areas of ICT expertise is exactly the same for every site globally. Please note that BIT/RC's view involves administrative applications from the business world, i.e. no robots, laboratories, neuronal connections, search systems or purely social media or sport-gear (smart-watch).

- The differences in the competence level of specialized engineers results from the use or lack of use of numerous exceptions for the implementation of purchased or leased software.

  - Moreover, most vendors sell their software according to their own insights (market share) and with far too many features in the hope of establishing a "de facto" standard, preferably with a "lock-in" for future updates (license fee).

- For applications that make virtual use of an infrastructure (cloud computing), the logical layout will remain the same but the tangible or physical components will become invisible: they cannot be presented to their users in a sufficiently detailed way. In such a case grey might be a more appropriate color than blue, red or green.

Hence, the drawing below summarizes the top level of any site in the current world of ICT for administrative purposes.

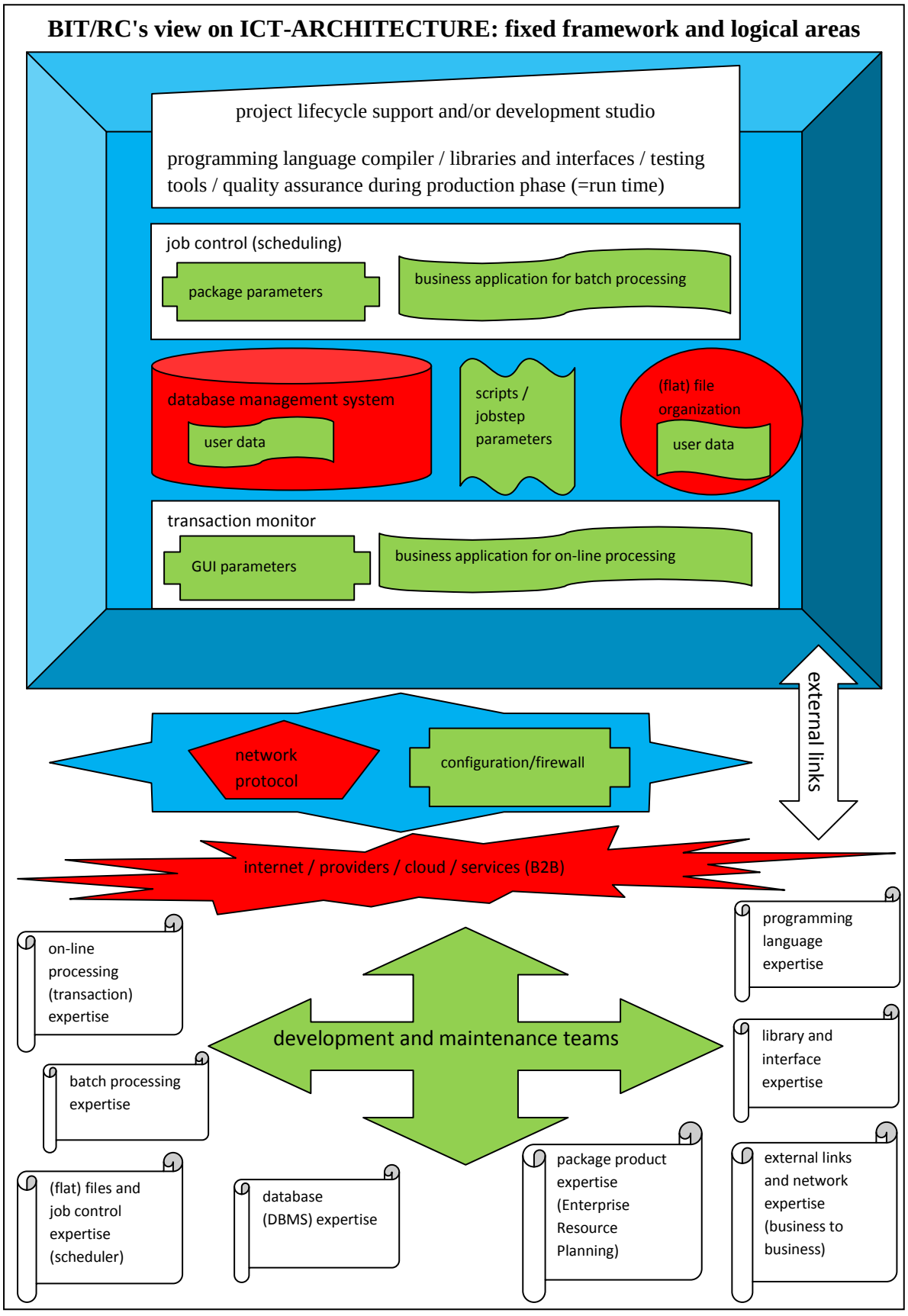
- In large companies the top level is repeated via networked systems, sometimes with different suppliers for basic elements and expertise.
    - The local network (LAN) or wide area (WAN) plus the internet show to a large extent where certain company details are processed within an organization. This is precisely why a cluster of interconnected systems is created and IT architecture can sometimes seem incomprehensible.
        - This is a problem that can be avoided at any time by mapping the top level of each system and location separately.

By zooming in on one of the colors in the drawing, a very complete layer is discovered that describes a company's architecture (and that is mainly the work of an ICT-architect).

- By completing the details (top-down approach) one obtains a picture of a site with the exact number of devices / OS, tools / software packages, applications / production jobs and parameters.
- By going into more detail, one obtains a more precise picture of the parts in which the business logic is locked up, as communicated by an SME to a business analyst and written in a document (the blue print analysis).
    - Wherever possible, the business do not call on a development team and the SME must set parameters for a software package (configurations / adaptations).
- The finer details give the image that the programmer sees on his development system (coding lines / IDE / visual studio).
    - Such a visible element of programming must also be overseen by an ICT architect in order to make the right choices when purchasing infrastructure (feasibility study).
        - The purchase of infrastructure on the one hand and the hiring of expertise on the other hand sometimes come together to start a large-scale project with the help of a SOW (statement of work).
        - In a SOW, in addition to tasks or activities (project scope), the project cost can either be agreed as fixed price or time and material. Consequently, all manner of legal conditions are covered, and the agreement of a SOW is in fact a very important contractual stage.
    - Occasionally the development takes place on an alien infrastructure (cross-platform / Java language) with only slight adjustments for the production phase (application porting.).

Consequently, with a reverse approach to ICT, in a " bottom-up approach ", it will become extremely difficult to understand the intent of the source code (reverse engineering) and even more difficult to structure the intermediate layers (libraries / interfaces).

- Throughout the major migration projects that I carried out, "bottom-up" often emerged as a means to check the inventory: for instance, are all artefacts that need to be retired or replaced fully mapped out?
    - As soon as it is discovered that a component is missing, there is a risk of extensive ad hoc repair work.
        - For that reason, I often plowed through the source code of applications (sampling): trying to find non-standard usage of a programming language or development in general.
        - My work as a migration-architect was facilitated by the well thought-out production of a REPOSITORY (a product that collects all source code for research), with special thanks to the Dutch company Cornerstone Technology (Dordrecht / The Netherlands).


BIT/RC's view on ICT-ARCHITECTURE: fixed framework and logical areas
project lifecycle support and/or development studio
programming language compiler / libraries and interfaces / testing tools / quality assurance during production phase (=run time)
job control (scheduling)
package parameters
business application for batch processing
database management system
user data
scripts / jobstep parameters
(flat) file organization
user data
transaction monitor
GUI parameters
business application for on-line processing
external links
network protocol
configuration/firewall
internet / providers / cloud / services (B2B)
on-line processing (transaction) expertise
batch processing expertise
(flat) files and job control expertise (scheduler)
development and maintenance teams
database (DBMS) expertise
package product expertise (Enterprise Resource Planning)
programming language expertise
library and interface expertise
external links and network expertise (business to business)

3.5. Guideline for required functionalities.

The 8 areas of expertise in an ICT-architecture are a good starting point for recognizing the required functionalities in applications: they are a guideline when a company wants to automate a manual procedure.

- The requirements of end users must fit into one or more areas for which an infrastructure has been set up using a fixed framework.
    - If this is not the case, then this causes a defect and it could lead to rejection of the automation or to the purchase of new equipment and software.

Below is a summary per area to allow subject matter experts (key person) to conduct an introductory discussion with specialized engineers (experts) before starting a new ICT project (business case / sponsoring / kick-off / project life cycle):

- **(1) The area of batch processing is for the back-office (for example: print-outs) that can consist of a central system or a cluster of midrange computers.**
- **(2) The area of files and jobs serves to produce data at a certain time (scheduling) and its security (backup) for eventual recovery (disaster center).**
- **(3) The area of databases serves for the storage of data (with emphasis on concurrent access / data integrity).**
- **(4) The area of on-line processing serves the front-office (mainly GUI / transactions), often with the help of local networks (LAN) that connect end-users to the central system.**
- **(5) The area of libraries or interfaces serves to reuse components in multiple applications (libraries / standards).**
- **(6) The area of global connections (the private WAN and the public internet) serves for the use of links with other companies (SOA services / B2B messages) or with independently operating devices (tablet, smartphone) that regularly synchronize with a central system (upload and download / file transfer) .**
- **(7) The area of programming languages and their integrated environment (development studio / programming framework) serves to determine the method of developing applications within a construction path (project road map / testing procedures / quality assurance and methodology).**
- **(8) The area of tailor-made packages (often an ERP) serves to produce a company's data without a construction path for programs within the company itself (implementation parameters / module configuration).**

After the introductory discussion, it can be decided to assess the feasibility of a solution within the company's ICT-architecture.

3.6. Trajectory for development of own applications.

The introductory discussion by logical area of the ICT architecture could possibly lead to the approval of an ICT project to develop the requested applications with a company's own resources (in-house development): the writing of general specifications (business analysis), detailed technical specifications (functions analysis), coding (programming) and test scenarios, which ends with commissioning.

- When outsourcing the development work, the methodology is always determined in advance. In house, a methodology will be imposed by an ICT architect in consultation with a quality manager. For example:
    - The methodology stipulates that the first document to be delivered from the new project is an action plan (first deliverable).
    - The plan of action divides a project into successive phases that start with a kick-off and end with a pause in order to control status of all activities (milestone, "go/no-go" decision).
    - The final document to be delivered in a project is an acceptance by end users.
    - After acceptance, the usage period starts with a regularly recurring maintenance version (production environment).
    - The expertise to retest parts of large systems (regression test) is part of the methodology (Quality Assurance / QA).
        - The knowledge needed to build systems seems unnecessary for QA at first sight. However, a QA-manager must be able to communicate with all kinds of experts (integration testing).

3.7. The profession of ICT architect.

An ICT architect will, by analogy with the construction world, first determine the broad lines of his fixed framework and only then fill in underlying details with the help of a very small team of engineers and data owners for various applications (key-users or SME's / blue print analysis document):

- His work starts with the design of the ICT infrastructure (5 basic elements) followed by a top-down analysis of solutions (logical areas), and the firm advice for new development that, after all of the test phases, must end with a final acceptance so that production can start flawlessly together with perpetual maintenance work (change management).

However, the ideal scenario rarely occurs for the simple reason that the majority of applications have grown historically without the professional help of an ICT architect (historically "mis-grown" or LEGACY may be better suited to very large ICT sites).

- For many companies it is almost impossible to return to the initial phase of "the blank status" (more than 500 man-years of development in a few years?). That is why an ICT architect usually focuses on new projects that must be integrated with an existing solution.
- For small companies, the moment arrives when they have to update their modern software and applications. If they skip it and stay with the older version (to save costs), then after a few years exactly the same problem will arise as for the big companies: the conversion has become unbridgeable (no ROI - return on investment).
    - Fortunately, there are vendors who provide tools (code generators) that effect the switch in an automatic way, i.e. including the syntax of a newer programming language, flat files conversion for batch processing, migration of data to an integrated database, re-use of ON-LINE transactions and their historical screen images.
    - Sometimes a company switches radically to a ready-made package (ERP software) rather that to start rewriting all of its outdated applications.
    - Often the preference for renewal is decentralization of data from one big system to several midrange computers (down sizing / Windows / Linux).

Therefore, some of the situations described above change almost everything about ICT within a company, and then it is a good time to call the ICT-architect.

3.8. Quality control and integration testing.

Striving for the highest quality (Quality Assurance) is a core task, of course, with numerous discussions about "how and when to test an application": such discussions can be summarized in 5 key words.

- ***(1) FLEXIBILITY .*** From the different starting positions to explain all areas of information technology, one subject will tick all of these conversation paths: that of quality and more specifically "the ability to adapt to new circumstances".
    - This flexibility issue is clearly demonstrated when the information in a company has to flow even faster than before, for the simple reason that changes in the world also affect the business information. For example:
        - If it takes too long to modify or add software, the cost price rises together with the cost to compete with competitors in the market (KPI key performance indicators).

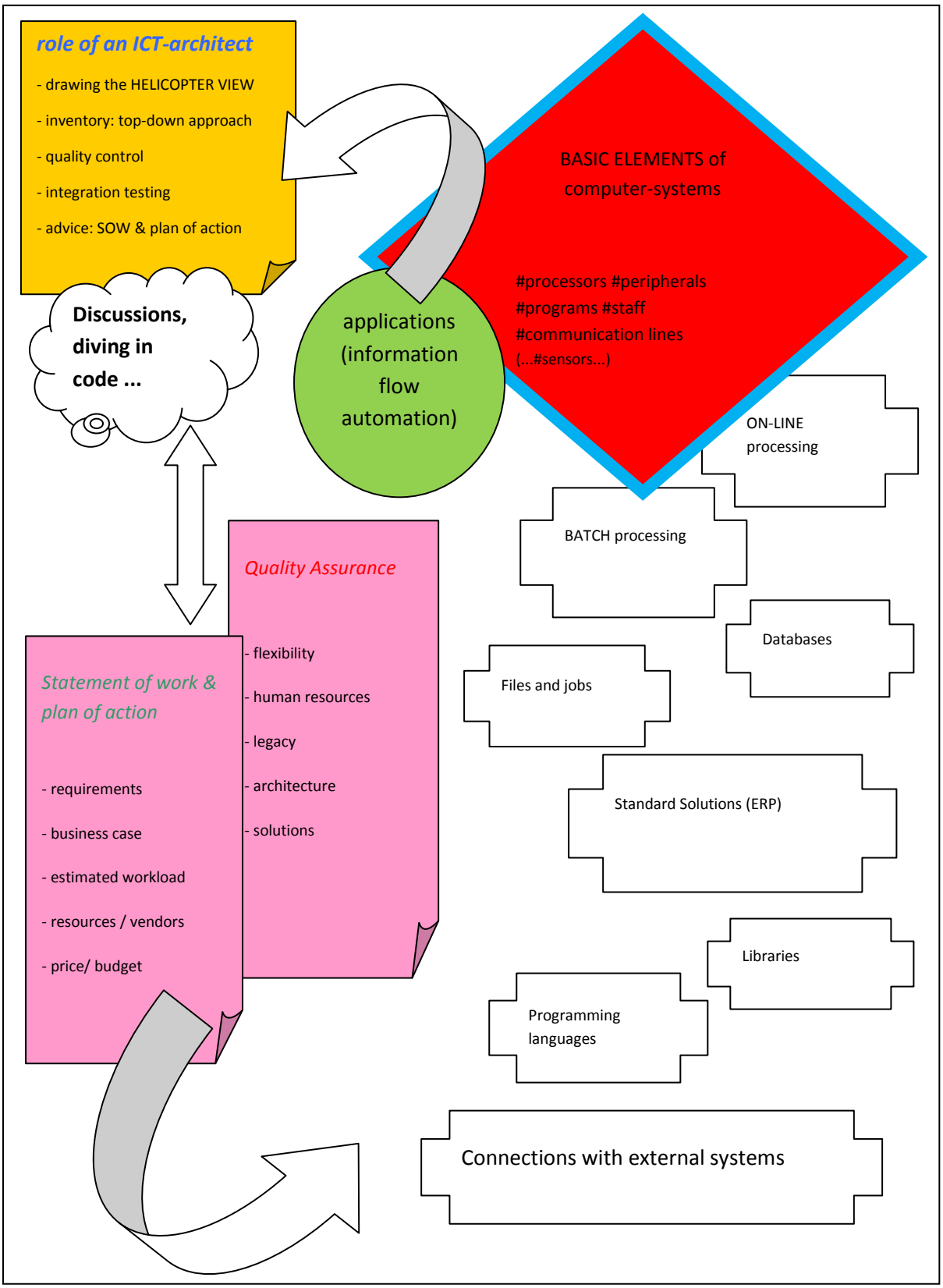
- The quality standard for flexibility also touches other topics of discussion such as safety, robustness, stability and return on investment (ROI): it explains why both background management and presentation, or operation in the foreground, need to be adjusted repeatedly without having to start from scratch.

- ***(2) HUMAN RESOURCES.*** Programmers are, as it were, totally unsuited to ensure the quality of information provision from the start of a project. However, in the initial period of the emergence of automated information processing no supplier was judged on its customer-unfriendly applications: they could refer to the technical inability, the lack of solid hardware and software that make flexibility possible.
  - The traditional approach of the past, with an awful lot of secrecy in the use of language, explains in many ways the image of today's implementers of automation as a slow responding club within the company of super fast businessmen.
    - The proliferation of PCs with very easy to handle software is in stark contrast to the standstill around very large systems with hard to understand infrastructure: that issue also plays a role in the communication between end users, managers and computer scientists.
  - The modern approach within companies aims at the maximum outsourcing of non-core tasks, which certainly includes the construction of software. Business knowledge (business rules), on the other hand, should stay within the company as much as possible.
    - This is an information dilemma that leads to many conflicts between people who determine the applications and people who build a computer-system and above all maintain it for as along as possible.

- ***(3) LEGACY.*** As the computer grew more efficient in terms of hardware and software, a profound shift arose to allow business applications to (a) first work with key-users from the "business" and (b) only at the very last stage call on "technical workmen" for writing the code.
  - By the turn of the century, the "all-rounders" no longer ruled over a legion of ignorant people, instead the breadth of IT knowledge spread over several areas of expertise: the power to take the necessary decisions shifted completely from the "techies" to the SME's.
  - It seems outdated today to have to wait months or years for a system that meets all the needs of a company, and yet "legacy", the state of the past, often makes the most use of a computer-system. For example:

        - By gaining some insight into the history of the computer in general and of the situation in each company in particular, more understanding can be gained by management as to why blindly imposed deadlines are not met.

- ***(4) ARCHITECTURE.*** A lack of basic knowledge of systems from the past appears to be the main cause of the eternal distrust between application-data owners and technicians who facilitate a large computer-system to produce the daily work.
    - Individuals who have to make very important decisions about brand new or outdated projects must therefore also have some basic knowledge.
        - Anyone who has never heard of BATCH will think that all information is available ON-LINE anywhere in the world without a jumble of purchased hardware and software, and with technical tools that remain completely invisible to the end users.
            - The understanding of a complex architecture via a number of simple principles, as they are always applied everywhere, is precisely the guideline that you will find in BIT/RC.
- ***(5) SOLUTIONS.*** The first part of BIT/RC starts from a historical perspective to acquire the necessary basic knowledge about basic elements of a computer-system: in order to understand what a company (the business) wants to realize in the future, one can best have a look at the solutions built a long time ago, and still running on that computer-system (sometimes dating from the 1960s).
    - In this way, everyone within their daily environment can also see that the ideal application is still searching for its ideal hardware / software: this determination makes it exciting for all of us to keep searching for the best and the fastest solution for automating business processes.
        - A sophisticated method of testing is required to ensure the quality of new solutions and their integration with older solutions.


role of an ICT-architect
- drawing the HELICOPTER VIEW
- inventory: top-down approach
- quality control
- integration testing
- advice: SOW & plan of action
Discussions, diving in code ...
BASIC ELEMENTS of computer-systems
#processors #peripherals
#programs #staff
#communication lines
(...#sensors...)
applications (information flow automation)
ON-LINE processing
BATCH processing
Databases
Files and jobs
Standard Solutions (ERP)
Libraries
Programming languages
Connections with external systems
Quality Assurance
- flexibility
- human resources
- legacy
- architecture
- solutions
Statement of work & plan of action
- requirements
- business case
- estimated workload
- resources / vendors
- price/ budget

4. Batch processing (console).

4.1. What is batch for?

Batch serves to process data on a computer without contact with end users because all the input has been bundled prior to the start and output is presented at the end of such a process.

- The distinction between batch and on-line processing only arose when the interaction between the machine and end-users (subject matter experts) was made possible.
    - In the early days of the use of computers for administrative tasks (business applications), there was never any direct contact between a data owner and a rotating program (electronic process).
    - Automation is called ELECTRONIC DATA PROCESSING (EDP). With the arrival of communication networks, EDP was replaced by the acronym ICT or IT, the EDP manager was promoted to CIO (chief information officer), and there was a proliferation of data-entry via smart input-devices (GUI-graphical user interface).
- As in the early years, today there are still computer processes that do not interact with end users.
    - The decision to use "batch" is realised during functional analysis, but is of course already assessed through the advice of the architect.
        - Batch processing is much faster and safer than on-line provided that all input data has already been verified (input validation).
- In the early years of the computer era, the human interface with a machine consisted of thousands of punch cards (80 bytes) which were pierced on a typewriter with holes in advance and then sequentially trimmed and mixed with other more permanent data (mother cards / sort & merge).
    - All punched cards are read neatly in succession as being a BATCH, which indicates a collection of all of the data from one day or an earlier time cycle.
        - The results of batch programs are often printed on hundreds of pages of paper (manual validation check) or transferred by the computer to newly punched cards that can be re-sorted and merged at a later date.

- Today, the batch interface consists simply of a text file (flat file) containing either Job Control Language (JCL / job and program parameters) or interpreted commands (operating system commands / script language).
    - A computer-operator is the person who starts the batch job, via JCL or script, and they receive notifications about the job on a screen (console), but they can not change the processing or the inputs except by taking the drastic step of aborting the job, for example: by a "kill" command via the console's keyboard.
- On large-scale systems (mainframes) in particular, there is a lot of daily processing (cyclic computer jobs) that functions like "a batch".
    - Usually such a batch-program reads input media sequentially. Data is collected from the business activities of a company a long time before the program is started and results are returned on output media solely after the program has ended.
        - For documents to be printed, a specialized computer-device is used that controls multiple printers (spooling). Other media, such as magnetic tape, was regarded at the time as the best place to save results in huge quantities over a long period of time.
    - Optionally, a database is used instead of files (flat files) to process a mass of data in rapid succession. Nevertheless, this remains part of a BATCH because the 3 phases (input, processing, output) will not be interrupted before the program effectively stops.
        - Once finished, a batch-program or job disappears completely from the computer-system in order to free all resources (especially temporary files and memory).
- In a production environment a file must sometimes be copied or deleted and then the OS (operating system) will provide a tool (utility program) to continue.
    - Such file manipulations, for instance a sorting and merging facility, are always part of BATCH PROCESSING. After all, with online processing, an end user has direct access to data, and in such circumstances perfect copying or sorting is not guaranteed to be safe because something can change elsewhere in the process (simultaneous access).
    - In a heterogeneous configuration of computers, the massive exchange of business data is a problem that cannot be solved with on-line processing because of the very long processor time required to handle large volumes. For example: with a data migration or with a data synchronization from the central database a job may take hours of computer-time (too short night)

4.2. How does batch work?

In most cases the Operating System (OS) will start a batch processing with a message (job started at <hh>: <mm>: <ss>) and come back at the end with a message that the processing completed as expected (normal program termination / job) or that processing was aborted due to an unexpected error (abort / abnormal end - abend).

- The way in which the OS accepts a batch via commands can be protected with a graphic element (GUI) as an intermediate layer: the instruction to start is given by pressing a "button" and the associated Job Control Language (mainframe JCL) or Shell script (Unix/Linux) is launched.
- In the case of a very fast batch program-run for which the order comes from a PC (desktop computer), the impression may arise that it is " real time processing ", but that is only apparent because the data is processed by a program that does not interact with end users: it interacts optionally via a console-device simulated on the PC.
    - One OS will use a command-text-line as an interface, for example in a shell-script, and another OS will read a job stream, for example in a text file with the outdated punch-card-format (the usual length of 80 characters).
        - The text lines follow a very specific syntax that differs from vendor to vendor (JCL statements).
    - JCL and script differ very much in many ways. For example: JCL does not support repetition (an inside loop), only a conditional statement is allowed (forward skip). With a script, looping is possible and assignments can be re-interpreted in flight.
- A design or functional analysis of batch processing must always take into account the simultaneous update of files elsewhere in the system.
    - With a database, a Data Base Management System (DBMS) takes care of such aspects (concurrent access), especially in combination with on-line processing.
    - With (flat) files, in some cases the OS ensures that the access for other jobs or programs is blocked until the end of the batch processing that was started with the same files as input or output (file allocation / job scheduling).
- Notifications about the start and end of a batch program are usually placed on a console near the mainframe-operator: something similar can be redirected to a file (output-redirection) so that the messages are retained, even for end users.

- Software can also output a report on the operator's console and even request input. For example: the COBOL programming language contains "ACCEPT FROM CONSOLE" and "DISPLAY ON CONSOLE" to interact with an operator.
  - This form of conversation is limited to the person who started the batch job and who still needs to send arguments to a program. For example: retrieving the system date (in many formats such as Julian-date for example) is a fixed recurring piece of logic in a batch program.
  - In COBOL a mainframe's argument can be read by using JCL from the job stream (SYSIN / system input).
  - Another special input-method, often used by scripts, is via environment variables that are set globally for a computer-site.

- The programmer who writes the code (source code) always starts with one main module (main program): that will contain the first instruction of the batch processing.
  - Sub-routines may then called from a library (either internal or proprietary).
    - After compiling, the modules are linked together (link phase) to for a run-unit for batch processing (executable program).
  - In some development environments, an entire "executable program" is wrapped to fit into a framework that itself takes up the task of "main program" (run-time environment).
    - An advantage of such a technique is the extensive tracking of each encountered instruction for "step by step" testing (debug mode). In the absence of this, the on-going status of a program could be displayed on the operator's console (debug messages).

4.3. Examples of batch.

Within the ICT architecture, the infrastructure of a mainframe is the most suitable place for batch processing. A server or midrange computer then follows as a preference.

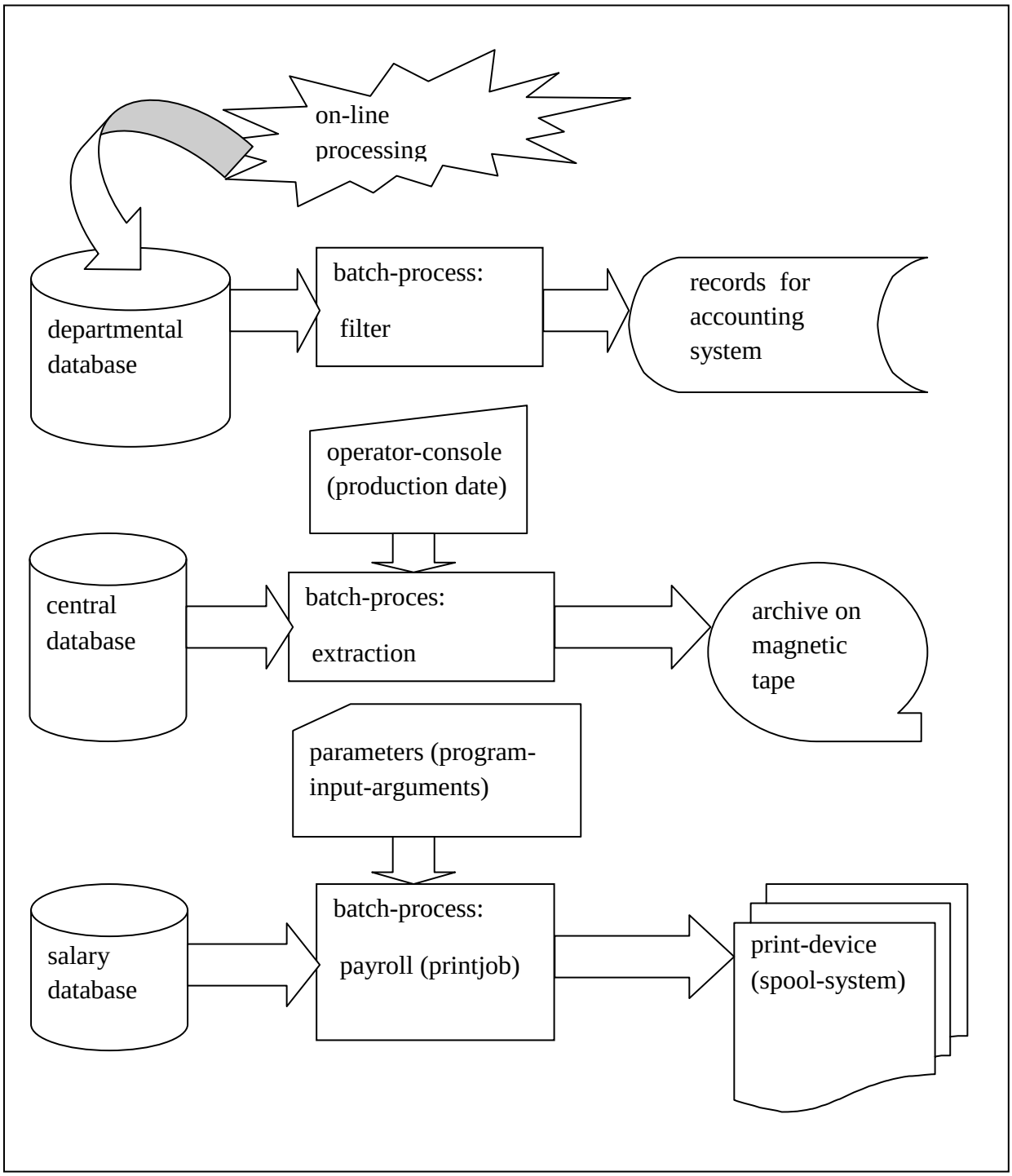
- A business department will, on a server (departmental computer), transfer its own data processing operations in batch and send results to a central computer: therefore, in a multinational or big company, the central computer is usually a mainframe.

- As you have probably realized, a desktop PC is not a suitable candidate for batch processing, since it serves only one user.

On a mainframe, server or midrange computer, the following batch processes are some of the most common:

- Cyclical filtering of data collected from on-line transactions (eg for accounting).
- Copying data onto external media for storage (eg archives on tape, network drive, DVD-media).
- The printing of centralized data (eg from payroll administration).

See below the schematic representation of some batch processing.



4.4. Search terms related to batch.

As a reminder: the tracing of encyclopedic details can be done with the help of the internet.

- Search terms are, in other parts of BIT/RC, usually denoted in brackets and/or in color, for example: ... (wiki-page) ...

For batch processing specifically, it is wise to start with the keyword "MAINFRAME" in combination with "BATCH PROCESSING".

- The most well-known mainframe manufacturer is "IBM" because of its marketshare of almost 90% built up over many years. Lesser-known vendors, such as AMDAHL and a few Japanese vendors, preferred to be compatible with IBM's software, particularly with regard to their tools (licenses for compilers, database engines and JCL-manipulation).
    - In France, "Bull" was an exception to the above rule with its own particular GCOS-series, just like "Siemens" that had its particular BS2000-operating system in Germany. In general they both switched to a UNIX-compatible OS which still exists today.

For an OS on a server system the most prevalent solution is Windows-based as it is better for connected PCs within a network.

- However, the Linux-operating system offers a free licensing agreement, and is therefore popular small companies. It often replaces older systems, for instance IBM’s AS400 family of midrange computers.

5. Files and computer jobs (scheduler).

5.1. What are files and computer jobs for?

Files are used for the electronic storage of data collected as a "batch". In most cases they are created according to a standard designed for their internal structure (eg they are the electronic media for photo, music, video, printer or just plain text).

- At the outset of the computer age there were no files: the results of batch processing were punched onto cards mechanically. Lists were printed to correct the non-valid input and then re-entered into the batch-program.
- With the arrival of magnetic tapes and mass storage devices (hard-disc) the rows of cards became known as "files": this is the digital method versus the mechanical one. The content remains the same in both methods, but the efficiency of the storage increases exponentially.
- During the functional analysis, the use of a file is usually assigned exclusivity to read or write it: it does not belong to all kinds of end users at the same time and must be allocated for a particular computer-job.
    - Since such exclusivity yields waiting times for other applications, it is preferable for a file to be processed by a batch program.
- Each file is designed for one type of data: this is called a "record type". In relational databases, later on, all file-records are replaced by rows (relational table).
    - Each record contains logical elements called "record fields": for each field a name has been devised that reflects the meaning of the content. For example: "CUSTNO" for a customer number, "ITEMNO" for an item number, "VATNO" for a VAT number, etc.
    - In a business analysis, this indication or suggestion of the meaning of data is recorded, and during the functional analysis each field gets its definitive place in the record so that all programmers can later work in an unambiguous way.
- To gain faster access to the contents of files, indexing was invented. For example:
    - a particular field is chosen per record, usually something numeric and valid as a unique key for the rest of the content in that record. All keys, when amalgamated, form a separate index file that is sorted in ascending or descending order, whilst each indexed key points to data in the unsorted file.

        - The access to a record is then done by first going through the index to find the location of the required record in the non-sorted file.

- Files still serve mainly to perform installations (download), or to save the contents of a central database (save / restore utility) or as additional information to a business-application (resource file).

Computer jobs serve to start batch processing. The way of defining and treating such jobs differs for each OS: Windows, Unix, a mainframe-OS all have their own ways to manage the flow of program execution from start to finish (job stream).

- A scheduler is a tool (utility program) which serves to guide the launch of computer jobs smoothly. For example:
    - The OS takes on as many jobs as possible when it has space in its memory. The scheduler, however, ensures that the constant flow of computer jobs takes place in the right order: at pre-scheduled hours or after it is certain that a previous job has been completed successfully.
- One manufacturer speaks of a batch job containing one or more "steps", the other one talks about "activities", "program units" or "commands": these all mean the same thing.
    - A "job step" corresponds to starting one process that is usually called an "executable program".
        - In turn, this process can start other processes (program units), but that happens without the knowledge of the scheduler. At the end of the step the scheduler learns how everything, went normal or abnormal, possibly with a more precise indication of what may have gone wrong (error code / abend).
    - For on-line processing, realetd batch processing must always be completed so that everything is ready to accommodate end users: it is the start of a monitoring program for connected clients that need resources like memory and disk space.

5.2. How do you work with files and computer jobs?

Files are still frequently used on mainframes and midrange-computers as a result of older applications (LEGACY). Programmers struggle with different formatting rules even when using the same vendor (proprietary software).

- For many years, files have been given a renewed look in a central database, for example:
    - the contents of a file corresponds to rows of a relational table.
        - Hence, a relational database can be migrated to another vendor easily.
- All computer-systems without exception have basic routines for reading and writing files on hard-discs or other media (device drivers).
- On a PC or small computer-system in general, programmers prefer to work with a standard notation like ".txt" or ".pdf" (file-name suffix): a lot of "free software" then hides the internal structure of a file and makes it easy for programmers to handle the user data in an application.
- The hiding of a file's structure is a common part of a third-generation language (instruction syntax - statement) whilst lower languages always handle this via an extra software-layer. For example:
    - a layer behind the programming language is formed by a library that provides the organization like grouping records in a larger buffer (blocking factor), an indexing mechanism or some type of compression (ZIP format).
        - A programmer uses the library with the aid of an interface with only the "user data" as a parameter.
    - A layer's interface can work in 2 ways depending on the program-compiler:
        - (1) embedded in the language, eg. with a READ verb in COBOL
        - (2) with a standard call mechanism to a library-function.

If a file is written, it is necessary to link it exclusively to 1 program: in the case of a mainframe this is done by a JCL-statement (job stream).

- The way to execute the link differs per OS, as well as the assignment (allocation / deallocation). For example: as soon as the processing starts, an indicator is set so that a file is not available for writing by another program.
- In a very few cases such an intervention is also necessary for a central database in order to prevent online-processing from nullifying the content for a batch-program.
- A mainframe usually links files via an assign-statement: such job-control-language syntax differs from manufacturer to manufacturer.
    - Programmers may use a graphical interface (GUI) for launching jobs without knowing the underlying JCL. For example:

        - through an ISPF panel on an IBM machine, a programmer can command the compilation of his just written source code. The results, with or without error messages, are sent to an output file that is requested by the programmer via another button on his ISPF panel .

    - All jobs on a mainframe are monitored by a team member from the production department (machine operator). They can intervene when something goes fundamentally wrong (program or system crash).

- A job stream other than JCL consists of a series of text-lines in a command-file: on a Unix system this is called a SHELL script , on PCs it is suffixed with .bat or .ps1.

    - Here too, GUIs usually exist that start the execution of a program without knowledge of the detailed content of the script.

- In one-off batch processing, the machine operator launches the job stream by a single command, for example "run" followed by typing in the name of the script or the JCL file. That can be very useful for a PC-user (software installation).

- In the case of a recurring batch programs at fixed times, the OS will assume an assignment (JCL or script) using a scheduler product (which usually has to be purchased separately on the software-market).

    - The scheduler operates with the OS in order to create a waiting list (job-queue) if a file is processed.

        - A good knowledge of all "batch" ensures that the mutual dependencies are respected with the aid of a scheduler: this is the responsibility of the production department of a computer-site.

        - A scheduler will deal with abnormal outcome by enabling people from the production department to make the necessary corrections before restarting a batch, optionally from a resumption point (backup / checkpoint or commitment / restore).

    - Another tool to send jobs that lead to job-launching or computer processes is a workflow product.

    - The supplier of a workflow package will always provide a graphical tool that helps to assemble and follow up the flow. For example:

        - an assignment is treated via a message sent by a client and received by a server. As a result, a workflow offers opportunities for jobs to be carried out on an external system (for example: a server is remotely accessible via the internet).

- A server can be fully set up with the only intention of accepting assignments using messages (XML-format) and returning the results via the internet (SOA-service oriented architecture).

5.3. Examples of files and computer jobs.

The OS determines how multiple files are collected in one folder: a full access path ("path-name") is always unique and serves to find a file on a computer. A remote file, optionally "in the cloud", requires additional information about the network-place.

- A mainframe always starts from a user name (system account). Next there is “tree"-information: branches pointing to underlying sub-folders. The use of computer-resources are limited per account, so no one can bring down the system by consuming all of the disc space.
    - The entire name for an access path is delimited with points or with a slash.
- On a Unix system, a folder is called a "directory".
    - Part of the "path name" serves as a starting point for extensions by a program that is started up and has no knowledge of it in advance (base directory). This information is stored in a special environment variable named PATH.
    - At the level of a directory it is possible, as for files, to impose an access prohibition for reading (r), writing (w), executing (e) or expanding with a sub-directory.
    - The first directory usually has no name but is sometimes known as "root" (referring to the start of all branches).
- Windows uses a drive-letter notation at the beginning of a "path-name": it points to the location of the "root" for a file or map (folder).

There are three main characteristics for the internal structure of a file: these are three types of contracts in a development environment that allow programmers to write code with the correct interface (API) when processing files.

- (1) The first characteristic is the file format imposed by the OS.
    - The file has a structure where multiple records of user-data are put into one control interval (data block) and each record is provided with control bytes (eg the length of user data, the location of the first byte within the block).

    - When programming, the layout of user-data must be closely followed, but information about the data block is very limited: the internal structure is handled via a library pertaining to the OS (basic I/O).
        - Sometimes user-data contains a different representation for digits (packed decimal) or a different method for decimal values (floating point).

- (2) the second characteristic is free classification.
    - The file is a sequence of bytes that form one long stream (data buffer). Programmers in C-language, or something similar, take into account a special sign to indicate the end of user-data in the stream (a binary zero).
    - During programming, bytes are written and read in a way that seems technically most appropriate (eg large buffers or small buffers / memory pool).
- (3) the third characteristic is a standardized classification.
    - The file is organized as agreed via an international meeting of vendors and this is respected by the software-builder who creates the particular file-type.
        - Under Windows, a suffix indicates which internal structure is applied. For example: .jpg stands for an image in a compressed format, .html stands for browser-statements, .pdf stands for a document with a format that is portable to any computer-system, .txt stands for "plain text" with lines ending with a special character (carriage return).

The data that belongs to end-users are either (a) divided into delimited fields of a record or (b) to be considered as a continuous text.

- (a) In the first case, an agreement is necessary between programmers to record the format via a record description during the analysis phase of a project. This organization-method enables the creation of an index-file: some fields are called record-keys because they are used to retrieve the location of records.
- (b) In the second case, it is possible to pass on the interpretation of the classification with marked fields. A standardized tool has recently been found for this organization-method in the form of XML (as a replacement for EDIFACT, an older standard)
- Each field in a record or each marked field consists of either numbers or letters.
    - Each alphabetical, numeric, punctuation or uppercase letter (upper case) is represented by one byte (character set).

        - In legacy systems numeric forms were usually compressed so that less bits were needed (packed decimal, binary, floating point are all representations of numbers that occupy less space than "1 byte per digit").
    - More recently there was an extension to 2 bytes per "character" if the set has more than the usual 26 letters of the alphabet (unicode) .

On a mainframe there still remains a job-scheduler that prevents resources being exhausted by a sudden influx of jobs (for example, 100 programmers simultaneously launching the compilation of their source code).

- An addition to the scheduler is useful for making company-specific selections for day, week, month or other recurring programs (job cycle).
- The equivalent on Unix systems are CRON and the AT command, on Windows there is also a task scheduler to start a planned activity.

Programmers can also utilize a technique to use an empty file as a marker (flag). For example:

- if the file does not exist then a batch program knows that it can continue and creates an empty file with a specific name (eg via a touch-command in Unix).
- Subsequent programs must wait until the touch-file has been deleted before they can start (remove-command).

5.4. Search terms related to files and computer jobs.

For computer jobs, the 2 most important search terms are: (1) Job Control Language (JCL) and (2) Shell script language.

For files in general, programming language manuals will always contain a chapter about the file organization that can be defined: it points to the program's interface with the internal structure that will be managed by the OS.

In order to know the structure of user-data down to the smallest details, it is advisable to execute a thorough analysis of applications. This kind of work is inevitable when migrating programs to another OS or hardware-platform: what would happen with files containing user-data? For example:

- When migrating, the user data is unloaded with tools from the system that created the data. The upload to the target system is realized with completely different software

present there. This analysis technique indicates clearly how file organizations differ or how fully compatible they are (synchronization).

- The unload results should be in the simplest structure of one text-line per file-record (flat file) and an upload will easily be able to recognize the layout of the plain-text.
  - The upload may result in an indexed file or in a table in a relational database: this depends on the migration goal. So, it is important to see the difference in the structure of data at the beginning.
  - Another example of using the analysis-technique is when a conversion to a different character set is required: this can have serious consequences for sorting user-data, especially on key-fields (eg when the sort / merge of numeric fields are mixed with alphabetic characters).

Creating an index, with key fields that are part of a record, has advantages and disadvantages: see ISAM on Windows or VSAM on IBM. For example:

- with an index, program execution is slower with a large number of additions in non-sequential sequence of key values.
- with an index, the random reading goes very quickly to a stable state (if there are no more additions).

Although PC home users will rarely or never need a scheduled computer job, a service program regularly informs them of certain tasks. For example:

- the "scan" of all files on the system detects viruses (malware) so that it can be removed. This is a good example of batch processing with files where the console-operator (PC-user) is informed about the status and a feasible action to delete the detected viruses.
  - Although this cyclic batch program has a button to interrupt the "scan", it is more of a "deadly shooting of the computer process" somewhere in the middle of it.

6. Databases (data definition).

6.1. What do databases serve?

A database serves to centralize all company data as if they were stored in one file containing all kinds of record types or media like photos, music, scanned documents: in addition to this main characteristic, a database is indispensable to quickly and efficiently manipulate a very large volume of data by thousands of people (CRUD : who does create , read, update and delete the user-data?).

During the functional analysis phase of a project, the CRUD-aspect is decisive for having a database set up by an ICT-engineer (Data Base Administrator).

- The presence of a database is clearly noticeable when launching computer jobs: where previously each used file was mentioned in a separate statement, one allocation is now of the complete "database".
- Without a database, on-line processing with hundreds of end-users slows everything down. All indexed files need to be managed simultaneously: an OS does not have any facilities at all for this requirement, so everything depends on the programmer's insight to anticipate a freeze of user-data during an update by another program or end-user. Even with a database, programs or users may experience a long wait time concerning the CRUD-aspect (deadly embrace).
- If several databases are in circulation then a company actually has a data warehouse (DWH): this term was initially intended to accommodate a diversity of manufacturers within one application with the main purpose of retrieving data (queries). Today, a DWH is managed by almost every DBMS (data base management system).

6.2. How do we work with databases?

A database can be used both in batch and on-line processing.

- A database is always managed with the help of a "package" (product): a DBMS is in fact a software tool under license of a vendor or free software foundation that can be present with many different OS's.
    - The package is never directly connected to the OS. However, an underlying layer (middleware) will still be composed of files that work together with the

OS, because of course disk space is an indispensable resource: this aspect is hidden from the business software, meaning that the programmer will not be concerned.

- An ICT-engineer (DBA) installs the package and manages it daily, primarily by analyzing statistics about the end-users. The DBA is the only person that creates new entities within this product, though usually at the request of a project team with key-users.
- Management by the DBA also includes other tasks such as taking backups and restoring the database.

- The main task of a DBMS is to preserve the integrity of data that is changed by a mass of end users at the same time: this is achieved primarily via on-line processing, but external users can also enter or change the contents in a database via synchronisation with the help of a network or the internet.

- Programmers of applications that use a database always use a programming interface that protects everything from physical read and write commands (the API for the DBMS). For example:
  - a programmer writes a statement that calls a sub-program with the necessary parameters to indicate exactly what to do within the logical fields: which row in a relational table should be read or written or deleted or updated (CRUD).
  - with the advent of an interface standard, there is the ability to keep an application unchanged when switching to another provider of a DBMS.
    - The first ever standard is called CODASYL with "DML" in COBOL and other languages (Data Manipulation Language), but another standard is more popular today: SQL or structured query language which is related to relational types of databases. For example: the DML will use the verb SELECT to read a relational table.
    - Particularly with the technology of EMBEDDED SQL , vendors of a program compiler have a trump card to uniformly record the business logic written by programmers (algorithm).
    - In the Java language, more specifically the J2EE framework, a uniform technique is also used for actions on the database (Java Persistence API).
  - An algorithm that is stored in the database itself, in turn of embedded SQL, provides the opposite effect for a uniform technology: such STORED

PROCEDURES differ in syntax for each DBMS. This means that they provide a difficult transition to a competitor (vendor lock-in).

- In a few cases, the DBMS is fully integrated with a package for on-line processing: that makes vendor lock-in even more intrusive.
  - The programmer starts from the tables in the database to create an image of it. The underlying logic of the package screens all manipulations, making it seem to the end user that it is directly connected to the relational tables.
  - The transition from such procedures to a different technique ensures that everything has to be rewritten from scratch.

6.3. Examples of databases.

The first databases were a collection of indexed files or files containing records that were connected to each other by disk addresses (pointers or database keys). The navigation through such databases is very fast and best suits on-line processing. For example:

- The location on disk is calculated on the basis of the pointer to a next or a previous record. The hierarchical structure, of a master-slave organisation or of an owner-member setting, can also be used immediately without the serial research of a mountain of key values.
  - This storage method leaves little room for logical changes (eg the database has to be reorganized when there is a new "record field" or one with a more extensive length).
    - Sometimes a company must choose an extended weekend to realize the database reorganization while there is a very limited service to end users (no network available).

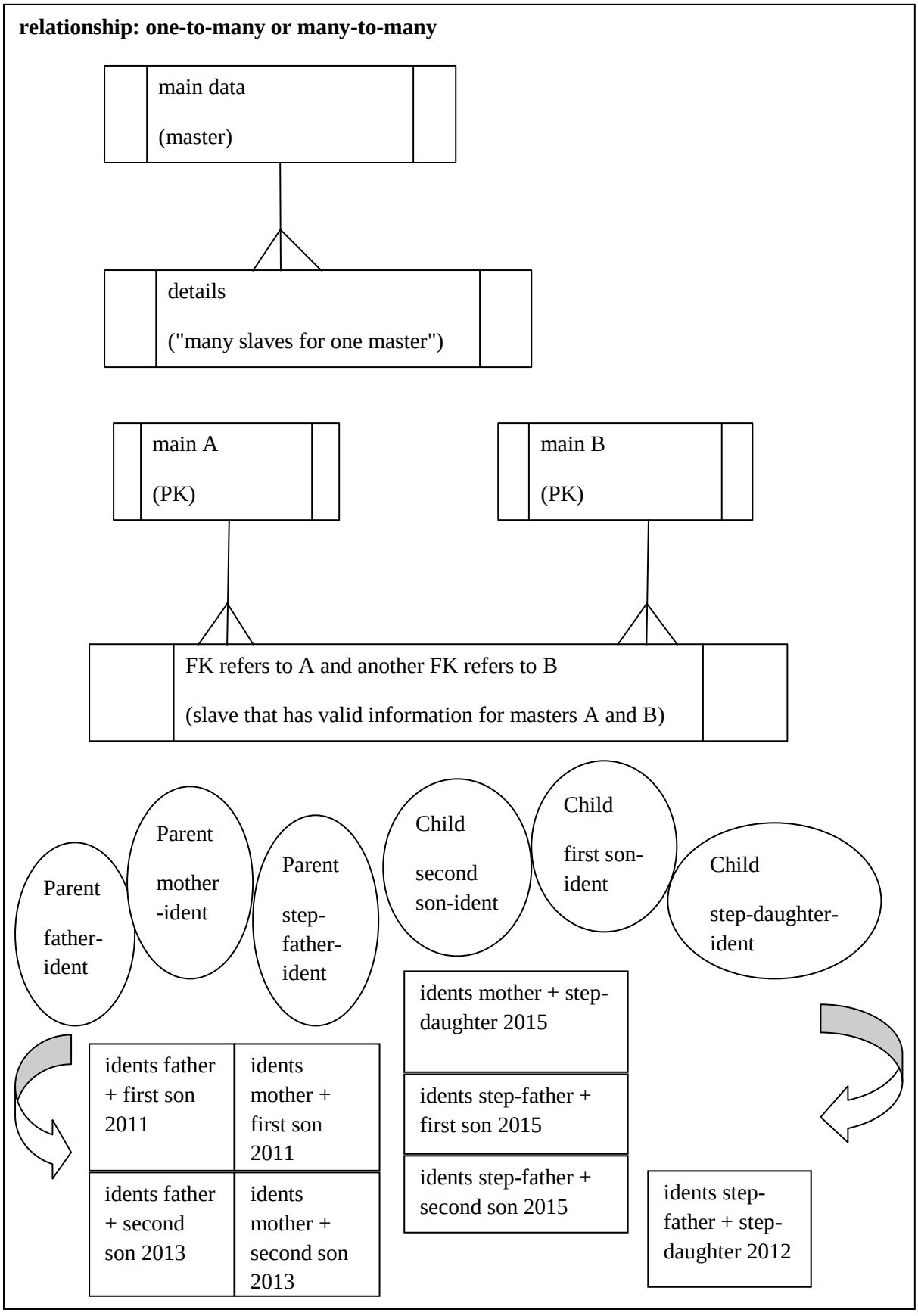
A modern DBMS uses the relational model to connect records (table rows): the hierarchical relations are logically defined instead of with physical pointers. For example:

- A table (relational table) with primary data about a customer's order has as a unique key: the sequence-number of the order (primary key or PK). The table with details about the order has several rows with a line sequence-number as the unique key: each such line refers to the primary data using the order number as a record field (foreign key or FK).

- Navigating through an order starts from the table with the primary data and continues with the table of detailed data, each time using the PK to start at the elementary row and the FK for rows containing order lines.
- An interface between a company's source code (its business logic) and the DBMS-package allows reading, writing, updating or deleting data with SQL statements.

- The relationships between entities, like an order and its order lines, (entity relationship) is divided into the following categories: "one to many" and "many to many".
  - A schematic representation shows which entity acts as "master" (primary data) by drawing a line from there into three lines under it.
    - An occurrence of primary data (one row) that is stored in the database will be related to one or more details (rows that refer to the primary one).
  - If there is a relationship between two entities whereby each act as a "master", then there must be an entity in between that refers to both occurring rows using an FK (foreign key) that suits both primary data.
    - The intermediate entity may contain information that is subordinate to both PKs. However, the goal remains to establish a unique connection between two main entities. For example:
    - A family has two sons (children) born in 2011 and 2013: after a divorce they will gain a stepfather in 2015 and also a step-sister born in 2012. This family situation will be recorded with "many to many" entities where, for example, a birth-date is recorded as a common characteristic (attribute) for both parent situations: the characteristic becomes the means for an identification of children with their birth-day as a unique key (PK) and referring to parents (FK).

relationship: one-to-many or many-to-many



6.4. Search terms related to databases.

In the case of mainframes, more specifically IBM, the choice is usually DB2 , on Unix systems Oracle is a leader and on Windows SQL Server is most often used. Other brand names continue to work with a significant but small percentage of the consumer-market although free software like MySQL is very popular in small companies.

- To gain knowledge about the set-up of a relational database, a theoretical study of data normalization is required (CODD - five normal forms). In this way there is often a gap between owners of the data and the people who understand SQL (structured query language) especially during the creation of tables (entity relationship).

Some db-vendors offer a programming-method to give faster access to relational data. For example:

- At ADO.NET a product running under Windows, large parts of the database are transferred by a web-server-program to a client (browser program). There the database-software works as if it were handling a spreadsheet (rows and columns).
- With MS / Access there is a direct relationship between entities of the relational database and screen fields.
    - To this end, the programmer uses a generic motor (business logic specific to the vendor) to which he adds his user-parameters that refine the functionalities built into the product (properties).

7. On-line processing (transactions).

7.1. What's on-line?

On-line processing aims to update user-data immediately as soon as it is entered anywhere in the computer-system. In many cases, during the business analysis end users can discuss how they want to display the input and output and many functional specifications can be added almost directly into an on-line program (GUI-painting / prototyping / agile computing).

- This way of DATA PROCESSING gives, in contrast to batch processing, the impression to end users that all of their actions on data are recorded immediately (real time processing), but in fact there is always a program that accepts or rejects the entered data (dialogue program).

The concept of on-line processing is therefore merely about on-line transactions: it can take a few seconds in order to examine and prepare data before it is stored in the computer-system.

- This means that there are intermediate steps to go from the start of a transaction to the end via business logic (conversation mode program).
- It is usually possible to have multiple transactions completed one after the other by the same end user: for safety reasons this is done via a log-in procedure with username and password, after which a long-term connection is established (on-line session start) until logging out (disconnection).
    - During a session, the end user often starts his actions via a selection of items (main menu items), for example: he chooses a commercial transaction and follows a dialogue with the computer-system.
    - Transactions sometimes need several seconds to complete a conversation with end users because data always needs to be checked for reliability (data validation). For example:
        - before changing some data the current state is shown (read). After entering the new state (update) the other side of the communication line will ask to confirm everything (either with the special "ENTER" key on a keyboard or with a "confirm-button" as a graphic object).
        - In this situation, the back-and-forth communication between devices (TCP / IP protocol) is only the physical bottom layer of on-line processing. The upper layer is the dialogue logic (transaction flow).

        - In the background of a huge computer-system, a very comprehensive program (TP-monitor) will constantly manage all transactions of thousands of users. Such on-line processing manipulates a company's data with the help of a database system, guaranteeing integrity for all kinds of simultaneous sessions (concurrent access).

The tendency to process everything online brings very high volumes of performance requirements: the number of transactions per second that the OS and TP monitor can handle.

- The correct estimation of traffic is an essential condition for dividing business processes between multiple computers (distributed computing) and the central archiving of commercial campaigns.

In the meantime, even long-standing batch processing usually continues unchanged (legacy system) despite more modern techniques.

- For complex applications with a very large number of programs (millions of lines of coding text) a company will be afraid to rewrite everything (from "scratch")
- Step-by-step adjustments sometimes take years because different bridges are being built (new interfaces between old and new applications).

As a result, on-line processing differs in many ways when building a new system rather than adapting an old system: choosing a newer solution often leads to a change in ICT-architecture and key-words to describe it. For example:

- The transaction is renamed into a question (request) and an answer (response) via a server-computer (service) that is connected to a central system.
    - Users are sometimes client programs that are written without any knowledge of the underlying operation of a service (Service Oriented Architecture).
        - Users of a service may not even belong to the company where the process is running. For example: for the direct exchange of supplier-data a bank may offer a service to accept an invoice and another service to present it to a customer.

7.2. How does on-line work?

The processing of data from countless end users and one central computer always happens as though one end user is the only one who uses the system (business application). The TP monitor provides enough memory space for simultaneous operation of sometimes thousands of end users (user- context) to deal quickly and efficiently with the available equipment.

- The Internet was initially intended for home users without the need to update any business data in principle: this form of on-line processing does not therefore provide the safeguarding of simultaneous updates in a database and usually only serves for enquiries (query system).
    - In principle, for such simple cases a TP monitor is not necessary: only a program that sends web pages (html) back from a basic computer (web server system).
- The communication protocol "http" , the sending back and forth of data between the end user and the on-line program, has the characteristic that after each action by an end-user the program acts exactly as though this is the first interaction.
    - Hence, the Internet-protocol is not suitable for a long conversation needed for on-line processing (session). Nevertheless, for companies there are many applications via the Internet that mimic a session and thus are equivalent to on-line processing (user's context data).

Imitating a session via the internet's www (world wide web), with the preservation of the user's context data, is done with the help of an application server (architectural framework) in collaboration with a web server that contains an http-protocol-interface (library-api).

- In contrast to batch processing, on-line applications (business logic) adapt to the specifications of a well chosen framework whether it is a TP-monitor suitable for a mainframe or an Internet Framework suitable for a mid-range computer. For example:
    - A framework ensures that data is kept separate from each client, ie that a context is created as if there is a lengthy conversation between a user and the central computer (session).
        - On the internet, a conversation is in principle impossible: it goes from one question via a browser-program (URL - Uniform Resource Locator) through to the delivery of a web-page (HTML) via a web-server.
        - Unlike a web server, an application server can in addition ensure that the origins of a few successive urls are remembered. A framework, such as J2EE or .NET, then provides an identifier for a long conversation that is very similar to a session, where a user logs in and out (eg an on-line application for a shopping basket with a payment function).
        - A TP-monitor provides the control of users logged on in a mainframe (sessions) and saves a context of data for each transaction separately. For example: from the demand for the start of a transaction via a

transaction-identifier (terminal device), followed by one or more screen dialogues (conversation program) through to the conclusion of the transaction with the definitive processing of data within a database (commitment).

- On a UNIX-system (Linux) a service usually provides a direct connection (socket) to a computer-system and stores a context of data per communication line. For example: from the first request by a client-program that is initiated by a user connected to the service, until a final update of a database that is performed by a server-program.

An on-line application, written in a programming language that fits with a network and framework, contains business logic to carry out commercial transactions. In most cases a company's goal with respect to on-line processing is to retrieve existing data, updating or deleting it, and optionally entering new information.

- An on-line transaction always follows a very simple procedure: (a) the user's question (client request) and (b) the computer's answer (server response) both in the order that is agreed with key-users (business analysis - flow chart). For example:
  - in a transaction there is a starting point and an end point that is set during the technical realization of the business logic (programming instructions).
- A transaction can also be seen as an input phase, a short processing phase, and an output phase that is completely finished around one subject, for example: it concerns only one customer and only one of his orders, etc.
  - If everything goes well, the program passes on the assignment to permanently store data on the database (database commit), if not it seeks to abort the processing (rollback operation).
- An end-user during on-line processing usually only wants to register once for a whole series of transactions per day (during one long session): the identification occurs in the morning (user login / password) and the session ends in the evening (logoff - logout).
  - With a mainframe, connections run via a dummy-terminal type (or via simulations on a PC thereof).
    - All on-line processing is maximized on the side of the central system, only the projection of a graphic form (GUI / form lay-out) takes place on the side of the end user (display / screen) with the input of data (keyboard / mouse clicks).

        - With more intelligent connection-types, options will arise to increase some checks on the reliability entered data: this creates a THICK CLIENT instead of a THIN CLIENT in terms of on-line processing.
        - Moreover, if there is a need to archive data locally (at the user's PC), the use of a browser-program is usually superceded by a customized-program with a fixed connection-type (socket programming).

On-line processing must always take place very securely if business data elements are changed immediately: the initial authorization to connect to an internal network authentication) is the most important part. For example:

- An on-line shopping application is done with a newly written web-program, but in the background the client-request is still converted into a transaction intended for a terminal-type connection with a TP monitor running on a mainframe.
    - The old terminal's data entry screen or form is therefore sometimes replaced by a web page: "screen scraping" keeps an identical view. Old software will still exist in the background and process the entered data.
        - In such an example, the log-in and log-out by the user remain with authentication checks according to old processes, in the same way as the specific control of a terminal device as if it were still directly connected to the central computer.

7.3. Examples of on-line.

With on-line processing, a programmer (source code) must always adapt some instructions to a framework: this framework in turn depends on the type of logical network connection available for end-users (network protocol - dialogue / conversation mode). The physical network should be an underlying layer that remains invisible for a company's program (business logic).

- If the physical connection runs directly from a central computer to a "thin client" (terminal device) without entering the public domain, that is without a telecom operator in between, then a central program must take on all aspects of the input-output handling: this main function is in fact a monitor to connect hundreds of users simultaneously.
    - The end user's actions are almost immediately forwarded to a company's routine that is started by the monitor. In each instance, the routine acts as if

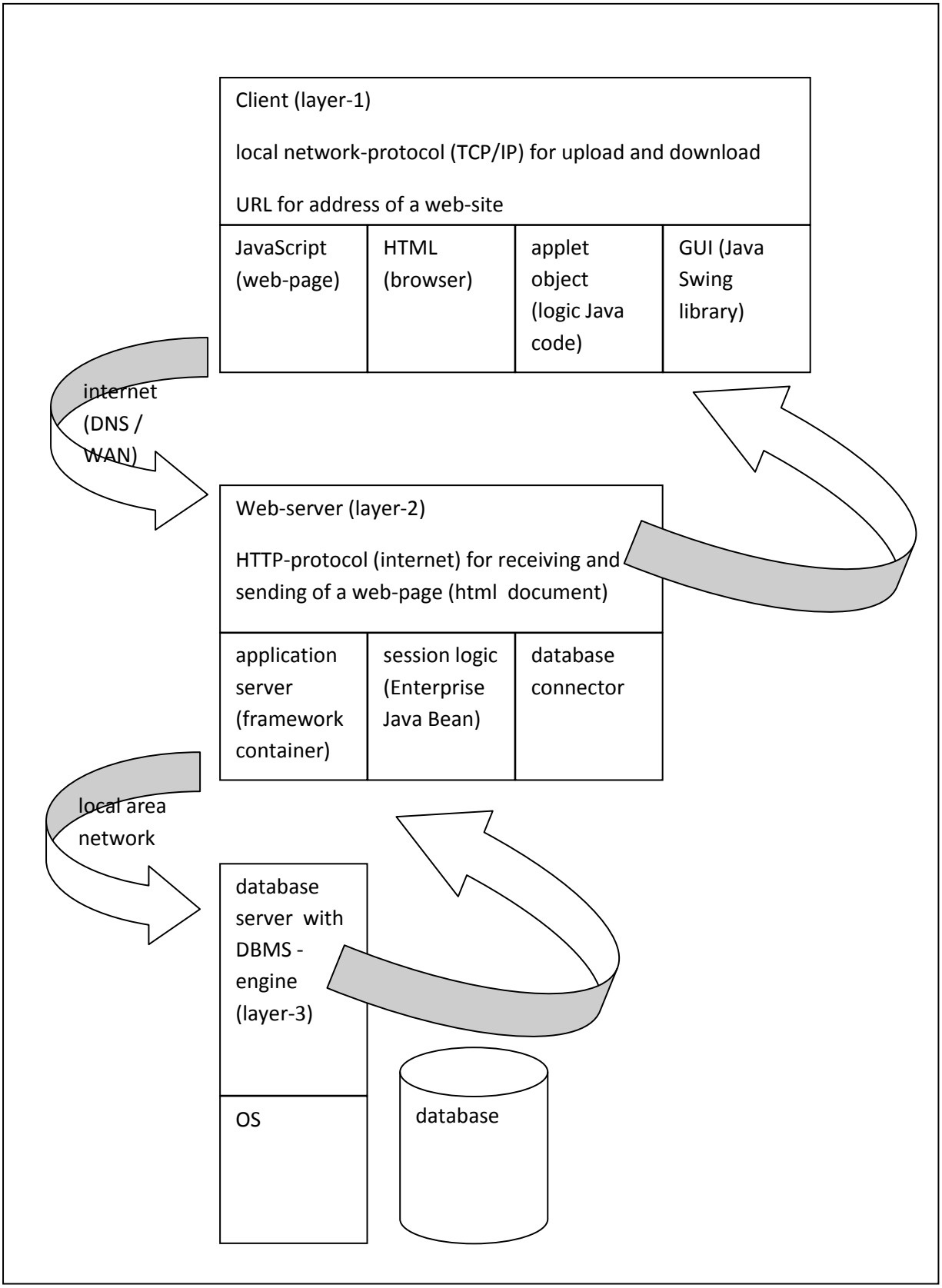
only one terminal is connected and manipulates data for one user at a time (business logic).

- If the connection is over a public domain, a telecom operator must be contacted with regard to some aspects of the protocol.
    - The end user's actions are processed by a client routine before being sent to the server side.
        - In many cases, an "intelligent" terminal is used for this purpose, which can exert control on input data (input validation) so that no unnecessary processing has to be done on the server side.
- If the connection over a public internet takes place, an independent domain manager must be contacted in addition to the telecom operator in order to establish a connection with a specific web site (www / domain name). In the field of data security, the communication protocol (http) between end users and specific web sites offers no protection against interception during transport via the telecom operator, which is an additional issue for a company.
    - The end-user actions are processed by a web server before being further sent to the central computer.
        - Usually, but not always, a "browser" is used as a client program to send the data to a web site via http, and then to receive data in the form of a web page created by a web server.
        - Such client programs may perform additional processing through instructions sent by the web server, together with graphical objects and other types of data in a language specially designed for a "browser" (HTML). For example:
            - a full web-page can be processed locally by the "browser" on the basis of html-settings, and smaller parts within with the help of JavaScript : this is a programming language that relies on the html-format of a web page and which the "browser" converts into a variety of actions on the computer screen, through keyboard input and clicking on objects.
        - An APPLET offers another variation to process data locally. For example: a web page contains a large object that consists only of instructions written in Java language, and which, together with graphical elements from a library (Swing - User Interface), takes a variety of actions on a separate part of the computer screen.

- On the server side, more specifically the application server, a container of components (for example in the form of JavaBeans software) provides the connection to a database. However, transmitting sensitive data back and forth should not just occur with the http protocol. For example:
    - adding a certificate during the transport of data enables the use of a secure communication via the internet: webshops etc. always use a slightly more extensive communication protocol (https instead of http).

The drawing below shows one possible arrangement of all components of on-line processing.

- Variants on this will relate to the programming language used or to the absence of an intermediate layer (eg 2-layer instead of an MVC model with 3 layers). There are also some variants for the location of a website. For example:
    - A URL is derived from what an end user types into the navigation bar of his web browser (website address and specific page with suffix .html or .asp or .php etc., possibly a default page, or an index page).


Client (layer-1)
local network-protocol (TCP/IP) for upload and download
URL for address of a web-site
JavaScript (web-page)
HTML (browser)
applet object (logic Java code)
GUI (Java Swing library)
internet (DNS / WAN)
Web-server (layer-2)
HTTP-protocol (internet) for receiving and sending of a web-page (html document)
application server (framework container)
session logic (Enterprise Java Bean)
database connector
local area network
database server with DBMS - engine (layer-3)
OS
database

7.4. Search terms related to on-line.

CICS on IBM is a monitor that only works with "thin clients": the kernel of the CICS program serves terminals that receive control codes embedded in a form display. Programmers have to learn these codes in order to anticipate the user's actions on their terminal.

- There are infrastructure additions around the periphery of CICS with specialized software (gateway) for a more extensive network connection.

The Java world requires a framework that is compatible with the J2EE agreements (standards). The equivalent in the Microsoft world is the .Net framework.

- A framework establishes the logic to send data from the client to the server and vice versa. Sometimes the programming language is specifically designed to add pieces of logic from business requirements.
    - Several programming languages are available for the framework of Microsoft ".Net" (eg C # or Visual Basic).
    - A framework usually has a rigid logic (runtime engine) to obtain the contents of a screen and process entered data. For example:
        - The (now obsolete) framework of Oracle "Forms" has to be supplemented with the programming language PL / SQL with imperative functions on both the client and server side.
        - The more modern framework of Microsoft "InfoPath" enables the programmer to create an "electronic form" and to work together with Office products or a database.

The default language for graphical elements on the client side in the case of a "browser" is HTML.

- A browser program on a PC only understands HTML. JavaScript can be embedded within an HTML document, acting as program elements in the form of object code.
    - Consequently, a web server only sends HTML as the "response" to a "request" (ie sending a URL using the http communication protocol to a domain administrator program somewhere in the world: this mechanism locates the web site where a server composes the html data according to instructions from a company (business logic).

- The use of a secure web site (https) is enabled through the help of a service company (certificate). This is an intermediate step before putting a site on-line: users will see "https" instead of "http" in front of the URL.

8. Libraries (interfaces).

8.1. What do libraries serve for?

A library serves to collect pieces of software that can be used by several programs: in contrast to sub-routines that remain an integral part of a program component, a library contains independent components for functions that a programmer does not have to write himself (software library file).

- The functions each have their call-name with which they can be linked to a program: in this way a batch process or an on-line transaction can access a specific part of the library and the programmer does not need to know all of the details (often not even the syntax of programming language).
- During the procurement of ICT infrastructure, external "libraries" are supplied (licensed software) that are part of a standard product or of the OS. During the development of company software, internal "libraries" are created that can be reused for later projects in-house (in-house development).
- Both the external and internal libraries are available to a project team for the development of new software. To be able to use them effectively there must be an interface from the programming language used by programmers to the call-name of a function (application programming interface / API). For example:
    - Suppliers of database systems (DBMS) provide an API to read and write data from a company's program using the syntax of SQL language.
    - For file processing, the API is usually tied to a specific programming language and the underlying OS.
        - The structure of a file is known by the programmer, but he never has to program the detailed functions for the storage of data. For example:
        - In COBOL, the API will use a READ to read a file, which results in a reading instruction for the OS that operates the hard disk (device driver) by the library attached thereto.
- A programmer therefore does not always have to re-write functions (business logic) that are also used by other programmers within his company: the only way to achieve this is by setting up an internal library.

- If a programmer uses a component, he must follow interface rules: these are the parameters that he gives during the call (call). The retrieval of results that the component has made are also subject to specifications with regard to their format (data type - language syntax).
- By building a library internally, a company can impose a discipline when manipulating data, which improves the readability of source code (maintainability).
  - During the functional analysis, the content of an internal library increases according to the number of times a part of the functionality is considered. The construction of a library starts with the technical realization but concentrates mainly on the maintenance of software from multiple projects, so that this expertise moves to the production department instead of a development team.
- If manufacturers use their own style to implement the components of their product, it is almost impossible for the programmer to have his source code work together unchanged with a similar product (non-standard API).
  - However, this migration disadvantage does not outweigh the complete rewriting of his own library for all kinds of functionalities, especially if they are very close to the OS, because such investment is almost certainly lost when switching to a new hardware platform.

- The use of a pre-compiler simplifies the work of programmers: the components of a library are referenced with the minimum of parameters, almost as though they were a standard instruction.
  - The real interface with a library, which can contain much more extensive parameters, is then completely hidden behind a very simple instruction: during compilation, pre-compiler statements are translated to the required API. For example:
    - With EMBEDDED SQL this advantage is very clear. The statement is universal, but the underlying hidden API is tailored to the database system.
    - For a "servlet", the PHP programming language generates all HTML and database encodings in the simplest way possible.
    - With on-line processing libraries, a pre-compiler is also very useful for writing complex communication assignments in the simplest way possible (no specific monitor-parameters).

- Some frameworks use short signs (annotations) to embed entire pieces of code without a programmer knowing the exact function. For example:
    - The development studio NetBeans manufactures all encodings for embedding a JavaBean in a J2EE application server.

8.2. How do you work with libraries?

Each OS has its own way of setting up a library. What a package means to an end user, is equivalent to the meaning of a library for the programmer: the technical characteristics do not have to be known to work with a library file (black box).

- To work with a library, the programmer only needs to know the interface: the IN and OUT parameters when calling a routine (eg INVOKE of a method or a CLASS).
- The interface between program-code and library-function is determined by the manufacturer of the called routine because that is where the rules are imposed.
    - Sometimes a library can only be used by a limited number of programming languages (interoperability). This problem can be solved by building a bridge (compatibility layer) or by writing a new frame around the source code (wrapper-routine).
- A library may contain components that do not comply with the principle of shared code: in that case, each call must be followed by making a copy in memory at the moment of execution. This adds an additional resource (memory) to the current program (run unit).
    - Some non-sharable libraries are so large that their merging into the run unit leads to problems with the available memory space of the OS at run-time.
        - This is solved by a technique in the OS: the use of virtual memory (paging). A programmer does not have to worry about this issue, but he must be aware of slower performance (elapsed time).

8.3. Examples of libraries.

Each OS has a set of libraries that are used by programmers during compilation or during batch processing. Usually this type of library is for file-handling, but it can also be aimed at a database. For example:

- For the use of a relational database there are rules around the insertion of SQL statements (Embedded SQL).

Every large company has a library in which recurring functions are collected. For example:

- The retrieval of the production date, the calculation of interest rates, the difference in days between two different dates, etc.

8.4. Search terms related to libraries.

A middleware product always has its library and frequently an API. Each OS has its own way of storing such a library on a computer system. For example:

- With Microsoft, a library has the suffix ".dll".
    - Due to proliferation and unclear agreements for implementation, especially for internet use, measures have been taken to provide certainty about the right choices (manifest / assembly).
- On a mainframe, library files are linked to a compiled program (object linkable format) and after the linking phase the result can be executed (executable program).
- Sometimes an executable program is itself called as if it were encapsulated in a library (run-time library). For example:
    - On a mainframe, an executable program may not itself be the subject of an EXEC- statement (JCL), but its name is passed on as a parameter to a generic main program, in the so called "run-time environment".
    - In the Java world the library gets the name "container". For example:
        - A web server manages a container with servlets that are written by the company's programmers: these program-components are started up by end-users via a URL: such location-information is obtained with the help of a "browser" and it indicates where a web page is produced with the help of a servlet (business logic).

9. Connections to external systems (network).

9.1. What are connections with external systems?

Connections to external systems serve to get input from computer systems of other companies or to send output to them (data transfer). They also serve to connect local devices from end-users in order to synchronize locally processed data with a central system (for instance via the internet).

- In the early days of EDP, one company printed its results on paper and handed it to another company that re-entered the data manually. Later on, data was transferred by handing over an electronic carrier (sending magnetic tape, disk, usb-stick by mail or courier): in such cases data did not have to be typed again, but problems could arise if a structure or the numerical format were not suitable for immediate processing at the receiving company. Such an issue could be handled by data-conversion software via batch-processing.

- After the deployment of external communication lines all over the world (backbone wired connection, public network telephone connection), a specific protocol for data exchange (ftp) was created. Today, the internet protocol (http) offers additional methods for adding a graphical interface that manages the transfer of a user's data (data exchange management).

- In contrast to on-line processing, connections to external systems are not intended to start and conclude an on-line transaction. For example:

    - a company's data is forwarded to another company without checking and following up the processing at the other side of the communication line.

- A connection to an external system does not allow registrations of individual end users: the goal is not to allow a massive log-in procedure to start a session between several clients and a server (mainframe or mid-range computer).

    - In contrast to on-line processing, the context of an end user is not saved, so there is no question of a thorough conversation (business logic).

        - Of course there is communication between 2 computer systems and that may be called a short-term session, but it is never the intention to process one transaction after another with many users at the same time (TP monitoring).

    - It is also not about filling in and processing "form by form" or "message by message" but about batch-oriented data, ie exchangeable files that, once

delivered, are completely managed by the recipient with their own computer system and ICT-architecture.

- The graphical interface (computer operation with screen forms) is aimed at following the status of an exchange between two computer systems. There are various ways to send data and get feedback information. For example:
    - An exchange of files is actually identical to a batch processing scenario where a file is copied. After the arrival of a file, the transfer session is closed or permission is given for a new shipment order.
    - An exchange of messages is either synchronous or asynchronous. For example:
        - an asynchronous exchange of messages between companies functions in the same way as internal E-mail traffic: it is processed if the recipient has time for it and in a way that can not be influenced by the sender.
    - In business transactions (E-commerce / B2B) sensitive data is sometimes exchanged with the help of a third party (financial institution). This requires a synchronous way of working: there is an immediate confirmation that the assignment has gone well or timed-out.

9.2. How do connections with external systems work?

With an external system belonging to another company, there must be clear agreements about the structure of the output (record lay-out) and about the communication line (network protocol). For example:

- Both sides must agree to an asynchronous connection or a synchronous one: this will have an impact on the duration of a transfer-operation and on sending messages back and forth.
    - On a mainframe, an external system can send a message by means of a message queue that works synchronously: the recipient immediately sends a confirmation that it is processing the data.
    - E-mail traffic is always asynchronous: the recipient does not have to be connected to the external system to process the message. The attached data is "dropped" into the computer system (mailbox).
- An international agreement provides the mechanism to provide or send data to any company (EDIFACT) because of the standard naming for record-fields.

- Since the advent of the internet, data has been packed using a universally known document format (XML). The structure of records is described inside the file sent to the other side of the communication line.
- With an SOA (service-oriented architecture) it is possible to call up a program on the other side of the communication line without knowledge of the internal operation or implementation. This virtually eliminates the need for an agreement between both parties of the data exchange.
    - A service directory is available for functions that are generally allowed to be used without any agreement (in the same way as the public "telephone book").
- Before securing a web-site against the interception of data by "pirates on the internet" (hackers), a certificate must be installed so that client and server can use https instead of http. Encryption of data during transport is guaranteed through the intervention of a service provider.
    - Networks in general contain multiple layers (layers) of which the upper comes into contact with the end user and the lower with the hardware. For example:
        - on the internet, http is the topmost layer that communicates with the browser that passes URL-information. The lowest layer deals with the cabling and routing of the packets of data from one server to another.

With the advent of smartphones, tablet computers and other variants of devices that can compete with a PC, there is a tendency to synchronize data. This is not the same as on-line processing. For example:

- The equipment of a company's employee who creates data (via a handheld device) and forwards this immediately to a business application on a central computer is an operation that must be treated in the same way as terminals within the architecture, rather than to be regarded as an external system.
- Synchronization on the other hand is an operation that deals with an external system in the architecture: the connection is established either following an agreement around security-rules or without such measures.

9.3. Examples of external systems.

External systems arise through agreements between companies or between individuals to exchange data. As a result, there are as many external systems as places where data can be

created: the agreement implies security mechanisms to establish a communication line between two computer systems. Ultimately a security measure is completely absent as in the case of many web-sites that only provide information to end-users.

- The processing of the entered data is entirely the responsibility of the receiving computer system.
- The preparation of data to be transferred is a task of the sending system.
- "File Transfer" (via communication protocol FTP) is the most used application in the business world for the transfer of data between computer systems.
    - If the receiving system is not managed by the company that sends the data, then there is, by definition, an external connection that goes beyond the scope of the architecture of the sending computer system.
    - If end users of a sending system fall under the responsibility of the recipient system then both systems must be managed as an integral part of a common architecture.
- On the Internet, a "download" of files is a popular form of using an external system to capture data. The ICT architect must provide test-facilities, in particular for the quality and optionally for new behavior within the site's architecture.
    - An "upload" puts the data manipulation in the hands of an external system. Here, the architecture of the sending computer system reaches its outer limit: it no longer has responsibility for data handling, except perhaps to guarantee the quality of a product for customers or end-users in general.

9.4. Search terms related to external systems.

An external system is connected by means of a network protocol that always partially runs via a provider, usually a telephone company.

- Security is an important aspect in a collaboration between different companies or individuals. For example:
    - Setting the firewall that allows FTP (eg opening a "port" number to allow a transmission of a file to a service program in another computer system).

In a company that employs home workers, there is actually no external system at all, but the security is at a very high level. For example:

- The connection of a PC to a central computer is made by means of a virtual private network (VPN). This is an external system that ultimately simulates that the end user has a direct connection: there is little or no added value in the graphic design and the software must be regarded as "on-line" processing with transactions directly on the central system.

In E-commerce (or business-to-business) the emphasis is on a faster and more efficient way of arranging invoicing data and / or payments between companies. For example:

- In an on-line shop (web shop) communication is not part of an architecture in terms of thousands of external systems that would be connected to a central computer: it is mainly a faster way for ordering products than by a telephone-conversation, fax or E-mailbox.

- Internationally, EDIFACT is a standardized agreement for the structure of data during an exchange between companies (through the uniform appointment of fields from a record).

- Between various systems that are perfectly aligned with each other in terms of software, a session with message queuing is the most efficient way for a data transfer (MQ series / IBM). This type of data exchange with an external system will be integrated into a company's architecture.

A service from a company on the internet might act as an intermediary step for exchanging data. For example:

- to forward a file from a PC everything is uploaded on the server of a service provider (cloud). With a URL (http: // ...) in an E-mail message, the recipient can "surf" to the provider's computer system and download the file from there.

10. Programming languages (development environment).

10.1. What do programming languages do?

A programming language serves to compose the logic for a computer process without the need to enter everything in machine-language. A processor of a computer system only ever operates with zeroes and ones: the machine-language.

- The machine-language is a set of instructions with just one purpose: to perform arithmetic operations. Their logic is implemented with the help of the binary system. Results are stored in memory and manipulated with the help of processor registers. For example:
    - a processor can read 4 bytes from a memory location and write them to another one: for this a 32-bit processor register is required for the manipulation of 8-bit characters (addressing mode).
    - Characters usually form a set of 255 variations that are agreed between manufacturers (standardization for a set with ascii, ebcdic, unicode).

A displacement of one hundred characters (bytes) in the computer's internal memory requires a 32-bit processor to execute 25 read-operations and 25 write-operations. With a programming language this can be completed with one command. For example, in COBOL (third-generation language), a programmer writes the verb MOVE.

- In addition to programming languages, there are also languages that initiate processes. For example:
    - JCL is a job-control language that is closely related to the way a mainframe or server starts a company's program (business logic). A shell script does the same for a UNIX system. The competencies necessary to master such languages are identical to those for "pure programming".
- Every higher level of a language makes it easier for a programmer by eliminating more of the underlying characteristics of a computer system. This abstraction level ensures that such a language is easier to learn than a language that is closer to the machine-language which is by definition the final version for every program running in a computer system.

A computer needs some basic software alongside the hardware that has no underlying interface: it is the core around which all other logic is wound (kernel / Operating System).

- The OS is software at the lowest level after the machine parts (physical hardware). Analogous to the construction world an OS acts as the foundation with a concrete floor slab on a piece of land. In principle, this is sufficient to place "something" on top that is completely free of shape or appearance (applications - business logic).
- Around the OS, a manufacturer builds the basic facilities for controlling a machine (device drivers / middleware). All other software uses these basic services usually in a layered form (API - library interfaces).
- An application (business logic) is the answer formulated by ICT-engineers to requirements formulated by SME-people (key users - user requirements): an "app" is an end result that will be divided by an ICT architect into one or more categories such as batch processing, file handling, computer jobs, database usage, on-line transactions and network connections with external systems.

- Programming languages, according to BIT/RC, serve to automate an administrative information stream using reasoning by a company's expert (workflow / procedures).
  - An administrative application consists of several parts (components) and makes as much use as possible of existing code stored in libraries.
    - It is rare that programmers have direct access to instruct the computer system: an application uses several intermediate layers, each of which translates an idea about automation or functionality further to a lower level (program compilation).

- A programmer would preferably write source code in a high programming language (source code) that is not associated directly with the operation of the machine or OS (device drivers - machine code).
  - If this is the case, the transition to another system means the loss of the largest part of the development investment, which in the long term leads to the complete rewriting of an application (in most cases this means 4 years of work for 10 programmers when millions of coding lines are involved).
  - The choice of a programming language can influence many decisions in a company and certainly the management of human resources. For example:
    - During an architecture study, it is always well researched whether a standard product (package) can be purchased that fits the programming language that a company already uses for previously written applications (legacy).

        - During the functional analysis, the choice of the programming language must be definitively determined so that any planned training can continue on time.

The programming work, including the testing of an application, always comes after analyzing the requirements of end users (user requirements).

- A conversation or reporting is seldom so formal that a computer immediately understands how the incoming information must be processed, archived and made available to end users in order to replace an information-flow with manual procedures.
    - During analysis, input, processing and output are separated to build a coherent and unambiguous application.
        - A programmer always respects the analysis and, if in doubt, will call on the analysis to formally describe the requirements again.
- In the future, the gap of understanding during a conversation with an end user and the programming work will decrease to such an extent that a robot will be able to apply all requirements directly. For example:
    - All intermediate steps from analysis of requirements to formal coding in a programming language become superfluous: this implies the use of artificial intelligence (AI).
        - The correct assessment of a context for multiple events or actions already seems feasible today for very specific machines that understand and execute orders from human beings without the intervention of an IT team.

10.2. How do you work with programming languages?

Each programming language, high or low level, always contains 2 essential parts: a part for verbs and a part for data-storage.

In support, high level languages provide easy descriptions for references to specific storage-media (system environment / file description / database definition).

- Verbs are always converted to machine-operations: these are instruction-codes for the processor together with addresses in memory where data is located.
    - Depending on the number of codes understood by a processor, the conversion will consist of only one per verb or several per verb.

- The conversion takes place either (a) prior to the execution of a program (language compiler) or (b) step by step during the run of a program itself (interpreted language).
- At the beginning of the computer age there were no compilers, so programmers almost always had to type in the machine's operation codes and addresses directly in order to create a program (symbolic assembly language).
    - Later there were libraries with a whole series of instructions that performed one or more functions. These functions are the result of the logical consideration of a programmer: if such abstraction leads to a higher level of complex processing with input and output devices (peripherals) then they are presented in a library (standard I/O functions).
- In programming languages one distinguishes the level of abstraction with the help of verbs and by access to the computer's memory: one language allows editing directly in a memory area, another defines only a virtual place.
- Due to the degree of abstraction, programming languages can be distinguished from each other but in fact they all have one major disadvantage: the programmer must have a good understanding of the ultimate function of verbs and data manipulation.
    - When an end-user explains items of a document to an ICT-engineer then a selection of variable contents is noted for processing by a machine: so, on the input side of a manual procedure there is usually a scanable document (text / image) and on the side of an automated procedure there is a formal layout of records in a file (with just a few variables).
    - The most obvious solution for non-programmers is of course the spoken language: say what the computer needs to do and then have it fully and automatically converted to operation codes and memory addresses that the machine understands. However, this might sound like a major step in program-development by an intelligent robot (AI).

Hence, programming is just as complex as mathematics for most people. It remains very difficult to instruct a machine in a completely unambiguous way and without the chance of misinterpretation. This way of working, just like with a mathematical formula, creates a broad gap between end users and programmers. For example:

- the elaboration of a series of consecutive instructions requires knowledge about repeated code (loops), tables of data (two-dimensial matrix), error messages (exceptions) and many more.

  - In addition, the techniques of programming have evolved into a way of representing the "real world" by classes. Older programmers have to learn and adapt their way of thinking out a solution. For example:
    - an object inside a program is encapsulated by a specific category such as a description of data-storage (private variables - public variables) and a description of associated routines (methods - property set).

- Moreover, the original data for an application, eg a scanable document, is separated into smaller pieces by functional analysts (record analysis) for the sake of being understood by programmers (record items). For example:
  - A distinction must always be made between numbers and other characters (variable type casting) because an arithmetic operation with letters makes the whole system fail (program or system crash).
  - The meaning (semantics) of data is stored in significantly abbreviated names: in a program this information is sometimes made completely unrecognizable in an algebraic way (logarithms), even if there is sufficient documentation about the logic behind the automated procedure (business logic).
- Although the source code of a program consists solely of written text, an ordinary word processor is almost never used for developing an application, except maybe on older types of mainframes where graphic tools are very limited.
  - To validate the syntax of the source code, programmers use a compiler-product.
  - More modern systems use a development studio (sdk - development kit / IDE-integrated development environment).
    - A studio is a framework for entering the textual instructions whilst at the same time checking the syntax and optionally compiling this into a language that is very close to the OS (object code).
    - Corresponding libraries are checked in time for the use of their interface rules.
    - The studio cooperates to the point that an application can be installed (product - application deployment) on the equipment for end-users (PC, handheld device, terminal, smartphone).

It is clearly the intention that the text written in a programming language ends with instructions to the machine (hardware / OS). There are two methods available for translation of the text:

- (1) converting the source code to machine language with the help of an assembler product oriented to a specific OS (language compiler).
    - A compiler is usually part of a package of libraries offered by a computer-manufacturer (license - free software).
- (2) converting the source code into a more compact version with the aid of an analysis product (language interpreter).
    - An interpreter works in the same way as an OS or kernel that reads computer-job-text (script or JCL): the compacted-code is examined and transformed into commands in machine language.
    - After each interpretation of a series of text lines, the OS executes the commands in machine language as usual with the help of libraries.

10.3. Examples of programming languages.

The oldest programming language is assembler with which an OS or kernel is formed.

- COBOL is known as the most widely used language and is a third generation language (3GL).
    - The advantage of this third-generation language lies in its transferability to another OS because the syntax is set by agreements between many computer-manufacturers (standardization committee).
    - The disadvantage of COBOL is that it is easily converted to assembler and then reworked into machine language that communicates directly with the OS: so, it generates differences between file handling by computer-systems.
- More recently, frameworks (SDK - software development kit) have been preferred as a development tool. For example: for Java, for C #, for Visual Basic, for C ++ and other object-oriented languages.

10.4. Search terms related to programming languages.

The builder of a translator-product for a programming language usually creates two documents:

- (1) a USER's GUIDE
    - the text in this manual corresponds to a form of course-material and shows many examples of program writing.
- (2) a REFERENCE GUIDE
    - topics in this manual are ordered in alphabetical order without regard to the best practice or usage.

For an IDE (integrated development environment) the choice is generally Eclipse (with Java in general), but NetBeans is quite popular (with J2EE for deployment on an application server rather than a local PC). There is also Visual Studio from Microsoft with languages for .Net (eg C #).

11. Standard products (packages).

11.1. What do standard products serve?

A standard product consists of one or more ready-to-use programs packed together with an installation product that any company or person may purchase and install on his computer system. In this way it serves for applications that a company or person is unable to develop or does not want to spend the time writing lines of code in a programming language. So, a package is a tool, a compiler, and an OS-program or other facility that runs on a computer system without any programming effort from the company or person who obtained such a standard product.

- In addition to these generic packages there are also numerous packages aimed at a very specific target group. For example:
    - A general ledger is kept up to date according to legal regulations and that will be cheaper and more efficient, per country or region, with a package offered by a supplier with thousands of customers, especially if the vendor is also responsible for maintenance work (update version - new release).
        - This is also the case for payroll administration, although here the preferred solution is for outsourcing: the import and export of information outside a computer-system managed by a company itself (service level agreement / SLA).
- A package that is organized in a modular way is able to offer a solution to many departments of a company and to centralize most of the information (no data duplication). For example:
    - A standard "Enterprise Resource Planning (ERP)" product integrates all of the information for all kinds of administrative applications: it is adapted via many parameters by key-users in order to customize the final solution.
        - In most cases a company has access to the ERP-code: this allows the perfect fit of the package with programs developed in-house (mostly via a special programming language or via an API-function in a library).

11.2. How do you work with standard products?

The most important considerations for working with a package are summarized during a feasibility study. For example:

- a mapping-report tests on a theoretical basis the business requirements of the functionalities offered by the supplier of a standard product.

A standard product must behave like any other application for which the computer system has been built: after testing and acceptance by key-users, and a hand-over to the production environment.

- End users must learn how to handle the product.
    - With a huge ERP package, the learning curve is rather high and specialization in a single module is recommended for the installation path (parameters set by a specific department within a company).

11.3. Examples of standard products.

The list of standard products is divided into three categories:

- Packages to replace in-house applications. For example:
    - An ERP package for payroll calculation, accounting, invoicing.
- Tools for the development of applications. For example:
    - Compilers for programming languages, a "studio" (IDE - development environment).
- Products related to the production of data by applications via a specific computer system. For example:
    - A "scheduler" to automatically start tasks and follow the computer's activity.

11.4. Search terms regarding packages.

The best-known ERP package is that of the SAP company (of German origin).

- Modules of this package focus primarily on companies with a mainframe or mid-range computer.

A vendor of standard products in general is most often linked to a manufacturer of computer-systems. For example:

- CA is the most common supplier on a mainframe from IBM.
- For databases (DBMS) Oracle is a well-known competitor of DB2 that is offered by IBM. For Windows, SQL Server is the logical choice for installing a relational "database" on a PC or mid-range computer.

Appendix.

The appendix to BIT/RC applies techniques in the world of ICT-engineering: so it is not an exercise in automating an administrative procedure or an information flow within a company.

- The first part (A1) briefly describes how discussions with end-users are picked up by ICT-engineers. While there are many themes, only a few are suitable for further development on a computer system.
    - Ultimately, ICT-engineers prefer to make a standard product rather than trying to solve one particular requirement for one particular end-user. The new software tries to cover as much as possible across many themes that have something in common: here it is named "the ABSOUV-package".
    - From the outset, considerations are made around strategic choices for a solution: this is what an ICT-architect must always do before the start of an ICT- project.
        - This type of work is similar to a feasibility study: it tells participants something about potential features and the estimated effort for further development (cost - time and material).
- The second part (A2) explains the ABSOUV-package from the point of view of intended end users. This description becomes the true starting point for the development effort: refer to Willy van Remortel, author of BIT/RC and inventor of the ABSOUV-package for more details.
    - Subsequent parts go deeper into the details, from general ICT-specifications to technical analyses and the programming of functionalities (in cascade-style from high-level to low-level).
        - The very first version of the ABSOUV-package is a prototype: it demonstrates the automated flow of information. Again, refer to the author for details of the source code.

Finally, sponsorship is needed in order to make a standard product for end-users, thereby formulating an answer on licenses, free software or any commercial action. Suggestions are always welcome.

A1. Practice: the ABSOUV- package.

A1.1. Themes.

The endpoint of the practical part of BIT/RC is determined by discussions between a few potential end users: their themes will be fit or unfit for automated processing of a specific type of information. For example: I was thinking a few years ago about collecting autobiographies or diaries (blogs) in a database that serves a scientific and social purpose.

- This theme arose from the digitization of photo material and stories about my family (I called it ABSOUV - my autobiographical souvenir). A kind of ABSOUV-database from people all over the world should provide a better insight into human behavior if the results are analyzed in a scientific way.
    - For that reason, I once passed a rough framework about the ABSOUV-theme to a professor at the University of Ghent who participates in the "human brain project" (HBP / European Commission): unfortunately my proposal was not considered suitable for HBP today. Allow me to explain the reason as follows:
        - from my philosophical point of view, a distinction is made between the physical characteristics of a brain and consciousness. Between science and religion a separation wall has been erected with on the one hand tangible proof and on the other hand an unmistakable faith.
        - Thus there is also a big difference in HBP's approach to discovering an "inner chemical" brain process versus the "perceivable appearance" behavior of people. Let's say: architecture versus applications.

Another theme revolves around social media, more specifically influencing behavior: the idea is to have a "virtual personal coach" throughout the day via the SMARTPHONE or PC. For example:

- how to give support to someone based upon their mood at any point of the daym? or according to their planned activities? or on the basis of their life course?
    - Today there is an "app" on the smartphone that already solves some of these requirements.

A1.2. Generalization and conclusion.

Actually, I am thinking about a solution for every theme that revolves around recognizing the meaning of a text in combination with other digital objects. Let me explain this in a simplified way:

- A digital text is an object that lends itself perfectly to dividing into pieces, technically speaking it is about "parsing" (recognize punctuations, words, sentences, paragraphs). Photographic material is a not ordinarily fit for "parsing", but it can always be provided with a description with the enumeration of specific characteristics (properties or keywords) and possibly be divided into fragments such as pixels (refer to the

identification of a person via biometric techniques). The same kind of recognition applies to music material that can be cut and pasted per note or per measure (refer to editor-software used by DJ's).

- In fact, all administrative information benefits from a computer program that recognizes attributes automatically and then presents the information in a particular form to end users, usually with arithmetic operations in between (business logic).

So, my conclusion so far is that the key word during development of a standard product must be: pattern recognition. That sounds like a technique of artificial intelligence, although in the context of BIT/RC's practice this must be regarded in a very modest way. Let us define two general principles of the ABSOUV-package at the starting point:

- (1) as soon as a digital object can be recognized, there is the opportunity for machine to transform it. To date, large computer systems store gathered data in a very compacted version and ultimately just a few programmers understand what really happens in detail with the information. In principle, ABSOUV does not touch the original object but still allows the processing at any time of the information as required by end-users.

- (2) a programming language is the most used "bridge-between-people-and-a-machine" in order to realize digital transformations. Making a kind of language accessible to end users who define for themselves the patterns as conditions for transformation is therefore the most ambitious principle of the ABSOUV-package.

A1.3. Objective of the software package.

Going deeper into a more technical discussion about ABSOUV, the package must serve to process digital objects for which the owner or end-user has devised a number of specific characteristics and has also created assignments or rules that lead to an automated process. The reasoning that goes with it reads as follows:

- Since the beginning of the computer age, fixed layouts have been used to enter data.
    - In fact, it is always about strictly structured texts that always contain two types for every piece (items):
        - (1) for the numerical value of a particle.
        - (2) for the storage format (compactness) in a file or database with limited lengths.
    - The definitions and assignments give a meaning to an item but one that remains very limited, namely, the possibility to process it as an amount or as text by a computer program. The much broader meaning, such as that given by a person who deals with computer data, is completely hidden in the jumble of instructions (logic) spread over countless components of an application with

computers. This creates a huge gap between man and machine instead of a language bridge. For example:

- The processing of all electronic data in the business administration is done almost exclusively with the help of programming languages that are far too complex for an end user (owner of the data): the person who knows the full meaning of data is left far behind once development of a program has started.
- As a result, only the ICT-engineer understands the internal structures of input and output on the computer system and a lot of time is lost due to the difficult communication with end users. For example: converting user requirements to functional specifications is a major step in automation.

- Combining various media has already become sufficiently common to the general public (see the internet), but there is almost always an informal way of data processing for mixed objects, ie there are still ICT-people needed to translate "the thinking of end users" into "a computer program that has no idea at all about the meaning of data items".

As a result, the ABSOUV-package will strive to allow end-users to realize all intermediate steps for transforming the objects they are familiar with, without knowledge of a programming language that is incomprehensible to them, and as far as possible focused on the meaning of data.

A1.4. Strategy for the development of the ABSOUV-package.

The package, based on the ABSOUV-theme, is developed in a few phases (milestones):

- a "plan of action" aims to provide estimates of tools and time required for writing the general specifications.
  - All general specifications are laid down as extensively as possible by the author of BIT/RC (keeping as such the intellectual property of ABSOUV).
- After general specifications have been written down, the estimates are adjusted for the rest of the development effort: functional analysis and programming.
- Trainees are allowed to make a technical analysis themselves (blue print): this yields more detailed specifications around a chosen theme that must meet the requirements of the practical part of the course (so, new general specifications may never arise without approval by Willy van Remortel).
  - An ICT architecture is drawn on the basis of the technical analysis, including the development environment.

A1.5. Expected bottlenecks of the ABSOUV-package.

Right from the start, the making of the ABSOUV-package appears to be a complex effort: there are several technical aspects that point to this.

(1) The first aspect is the application of extensive PARSING techniques to digital objects (many sources of data or media):

- the parsing is in the function of pattern recognition, for example: keywords within a free text in general, or recognition of a person's face on a photo.

(2) The second aspect is the application of GENERATOR techniques:

- a generator ensures the automatic creation of code-lines, for example: based on rules from end users, TEMPLATES are filled in for the processing of several sources of data.

(3) The following aspect is the application of SECURITY techniques:

- this embraces a strict shielding of personal information for the uninitiated: details about someone's privacy could be found within sources of data.

(4) The following aspect is the application of techniques for a generic DATA-MODEL:

- this allows an end-user to build up any file-system or database according to his line of thought without the intervention of ICT-engineers (except maybe for the initial setup on a computer system).

(5) The last aspect is the application of PERFORMANCE techniques:

- the processing time must be as short as possible because the sources of data may come from all over the world to a central system (via the internet). Pattern recognition, with millions of relationships in a file-system or database as a result, should also happen via a very short-term method.

A1.6. All-round aspects of a standard product.

There exists a minimal gradation of "generic code" within a computer program: this is observed from the early days of electronic data processing (EDP). For example:

- it is fundamental that input is always processed without knowing the exact content in advance, ie the user's data is obviously created a long time after the software is delivered to end-users.
    - Consequently, a program must always be able to accept or reject input. In some cases, the entire computer process sometimes stops with a fatal error:

avoidance or repair are inevitable issues in the world of ICT (crash analysis - disaster recovery).

- This observation of all-round aspects can grow further in its abstraction and thus reach an absolute height: a program can not perform everything an end-user wishes to do.
    - The boundary of generic behavior by a program is reached when the internal data-structure is chosen together with allowed formats of digital objects (input / output media). At this point, end-users are confronted with a shortage of knowledge about data manipulation.
        - Programmers need firm agreements (standards) concerning the layout of texts, photos, films, music. End users, or their individual projects, only agree about the content of their data. However, program code is not concerned about the meaning of data.
- The ultimate goal of generic software must be to make the barriers between end-user and programmer as superfluous as possible.

The ABSOUV-package is generic software: it enables any object that was conceived by an end user to be recognized, searched, analyzed and possibly converted to a new object.

- Such a transformation is done with a pre-written scenario that is recognizable for an end user (owner of the data).
    - This leads to a product that requires a new way of communication with a computer system, very similar to a programming language, except that it will be one that should be perfectly understood by end users.
- For a series of digital objects of an end user, a line of thought that is viewed from the top is regarded as an abstraction of the content to be processed. For example:
    - for every abstraction there are underlying characteristics that support the line of thought as "pillars under a theme".

A1.7. Transformations and patterns.

The greatest added value of the ABSOUV- package is without doubt the development of a new language as an interface for an end user to manipulate objects from his own environment without having to obtain a thorough knowledge of programming.

- The owner of data keeps a description of physical objects in his mind and he knows the patterns that occur in them.
    - Patterns are just like the pieces of a drawing that, when brought together, form a recognizable figure. Sometimes, not all pieces are required for a complete assembly.

- In this way, keywords within a digital object form a recognizable pattern, just like the properties that are assigned to an object. Every pattern must fit perfectly into one line of thought: the abstraction of physical objects as an equivalent of their digital representation (text, photo, music and many other sources of data).
    - The storage of patterns in a file-system or database will allow the end-user to approach any digital object from one or more abstractions.

- The idea to transform sources of data starts with an abstraction and proceeds with pattern recognition based on characteristics found in digital objects.
    - This means either a new state of a digital object or a complete conversion to a new object, or even to another abstraction with other sources of data.

A1.8. Commercial exploitation.

From the perspective of commercial exploitation, a clear distinction must be made between formal and informal data for input and output of the software package.

- For example, a transformation cuts and pastes photographs from the written press, the accompanying activities of another person or a pet to the biography of a participating object.
    - The transformation takes place by means of successive steps, just like in a film-script (scenario).
    - Consequently, the ABSOUV-package can also serve to read and convert administrative documents or forms with the help of rules.
        - In this context, it should be emphasized that digital forms are formally described, just like for the development of programs that automate an information flow within a company.
        - A form must be suitable for automation and therefore this practice does not correspond to free texts and other media that are not bound by formal rules or a specific syntax for their content.

- The ABSOUV-package can possibly serve for all kinds of techniques related to non-structured data, for example: big-data analysis, artificial intelligence, rule-based system, search engines.
    - Moreover, the connection with event-driven software and real-time processing (such as CEP) could provide a boost for users of robotisation or data streaming. For example:
        - analyzing unopened E-mail, including their attachments.
        - Filtering logs, music broadcasts, etc.

- Plowing a web site's data in search of specific information (pattern search).

The rest of the appendix is under construction: here are the topics.

A2. Basic principles.

A2.1. Basic term: abstraction.

A2.2. Sub-model, main pillar and sub-pillar.

A2.3. Associated pillar.

A2.4. Pattern recognition, indirect link.

A2.5. Associations and format mask.

A2.6. Scenario text, generator and transformer.

A2.7. Feedback and archiving.

A2.8. Overview of the ABSOUV-package in drawings.

A3. Specifications in general.

A3.1. The flow of data.

A3.2. Data at its source and results.

A3.3. A (new) language.

A3.4. Participating objects.

A3.5. Characteristics.

A3.6. Eentities.

A4. Functional analysis of the ABSOUV-package.

A4.1. General guidelines.

A4.2. Use case: enter abstraction.

A4.3. Use case: enter key characteristics.

A4.4. Use case: enter associative characteristics.

A4.5. Use case: entering associations and sub-pillars.

A4.6. Use case: enter pattern recognition.

A4.7. Use case: enter transformation scenario.

A4.8. Use case: enter indirect link.

A4.9. Use case: enter the start of the computer process.

A4.10. Use case: consult feedback.

A4.11. User manual (implementation of the ABSOUV-package).

A5. Details about the ABSOUV-language.

A5.1. Definition.

A5.2. Basic elements.

A5.3. Summary ABSOUV-language.

A5.4. Typical process flow (business logic).

A5.5. SECTION TRANSFORM.

A5.6.SECTION PROPTERTY.

A5.7.SECTION RULE.

A5.8. INSTRUCTION REPEAT.

A5.9. INSTRUCTION SWITCH.

A5.10. INSTRUCTION CALL.

A5.11. INSTRUCTION LOG.

A5.12. INSTRUCTION GENERATE.

A5.13. LOAD.

A5.14. RELOAD.

A5.15. CHECKPOINT.

A5.16. RESTART.

A5.17. COMMIT.

A5.18. Cutting and pasting.

A6. Migration of old applications (legacy).

A6.1. Goal and method.

A6.2. Traditional method.

A6.3. New method.

A6.4. Conversion and migration.

A6.5. Legacy declarations.

A7. Standard objects (ABSOUV-library).

A7.1. Annual calendar (AbsF200).

A8. Artificial intelligence (AI).

A8.1. Optical recognition of characters (OCR).

A8.2. Voice technology, facial recognition (biometrics).