Business & Computer Science

'It is not from the benevolence of the butcher, the brewer, or the baker that we expect our dinner, but from their regard to their own interest.'

The Wealth of Nations, Books 1-3
Adam Smith

Dario Loi, Student, Department of Computer Science
Applied Computer Science & Artificial Intelligence
`loi.1940849@studenti.uniroma1.it`

Contents

1 Course Introduction

1.1 Course Aims

Despite what the name might make you think, the course is *still* a practical one, it aims to introduce how computers work in a real world scenario, such as a company.

The Purpose of a Computer Through the lens of this course, the purpose of a software is to improve the company's efficency and throughput, everything is evaluated through a framework of costs and profits, hence, even something as simple as buying a pen must be evaluated at multiple levels, from the cost of the pen itself, to the cost of the time spent by the employee to go to the store and buy it.

1.2 Course Structure

The course is usually organized in two parts, alternating on a weekly basis:

Day	*Purpose*
Wednsday	Lecture
Thursday	Seminars

Table 1.1: Course Structure

Naturally, the seminars are *included* in the course's materials and are hence expected

The exam is also split into multiple sections:

Part	Contents
Written	30 to 50 closed questions
Oral	Optional, to raise marks

Table 1.2: Exam Structure

The general information about the course is available at the official course page.

1.3 Information Systems

As studied in the first unit of *Data Management & Analysis*, an information system is a set of components that allows an organization to collect, process, store and distribute information, it is *not* necessarily something software-based, it can even be a file cabinet!

Definition 1.3.1. *An Information System is something that* Manages *the flow of information in an organization.*

Information Naturally, information is something *abstract* and *immaterial*, it is naturally difficult to manage, but it is possible to construct a set of rules, conventions and tools that allow us to represent it in a way that is *manageable*, this process is the *Flow*, and hence, as defined above, these systems are called *Information Systems*.

Computers When a computer is involved, the Information System is naturally labeled a Computer Information System, or *CIS* for short.

This chapter aims to give us the following capabilities

- Define what an Information System is
- Describe the history of Information Systems
- Describe the basic argument behind the article *Does IT matter?* by Nicholas Carr.

An information system is made up of *Three* main components:

- People: In charge of decision making and organization
- Technology: Hardware and Software that supports the business
- Processes: Collecting and storing information

In General This is a pretty broad definition, that allows a wide variety of software to be seen through the lens of an Information System, from network analyzers, to national healthcare systems, to online education platforms.

Acronyms Depending on the application, an Information System can be called by different acronyms, such as:

- **MRP**: *Manufacturing Resource Planning*
- **CIM**: *Computer Integrated Manufacturing*
- **SAP**: *Systems, Applications and Products*

1.3.1 Anthony's Triangle

Anthony's triangle is a diagram that categorizes the three purposes of an Information System:

1. Strategic (*Executive Information System*): for senior management decisions
2. Tactical (*Management Information System*): For middle management decisions
3. Operational (*Transaction Processing Systems*): For daily transactions of business

1.3.2 Information System Components

Several Components work together to add value to an organization, they are

1. Hardware: The physical components of the system
2. Software: The programs that run on the hardware, they can be:
   - Operating Systems: The programs that manage the hardware
   - Application Software: The programs that are used by the users
3. Data: The information that is stored in the system
4. People: The users of the systems, both producers and consumers of infomation
5. Processes: The procedures that are used to collect, process and store information[1]

This is somewhat redundant to what we stated beforehand in definition 1.3.1, but it is important to restate it to emphasize the fact that the system is composed of both *People* and *Technology.*

1.3.3 Processes

One of the most important components of an Information System is the *Process*, it is the goal of the Computer Information System to optimize the processes of an organization, bringing about an increase in efficency.

[1] Organizing something in *processes* that is, a series of well-defined steps, brings about a series of benefits in productivity.

Processes Formally Since processes are an advantageous way to organize work, it is important to spend some time to also give a formal definition of their nature.

Definition 1.3.2. *A process is a series of well-defined steps that are used to achieve a specific goal, it can be defined as a set of* activities

1.4 Does IT Matter?

Nicholas Carr, in *Harvard Business Review*, argues that Information Technology is not an *Investment*, but rather a commodity, so something that must be managed to optimize the company's profits by reducing its operational costs.

IT as a Marketing Tool It is also interesting to note how a company is percieved as better when it uses IT, and when this IT is of high quality, therefore IT can be seen as a *Marketing Tool* that can be used to attract customers.

1.5 Mainframes

A Mainframe is a class of computer that is usuall used as the heart of an Information System, where everything is *centralized*, as opposed to a distributed systems, users, which in this architecture are defined *dumb*, are not allowed to access the system directly, but rather act as consumers of its services as allowed by the system's administrators and operating system.

Definition 1.5.1. *A* Mainframe *is an architecture in which a central computer with very high processing power is connected to a multitude of* terminals *through a* star *topology, where the* central computer *is the* hub *of the network.*

Even though *Mainframes* are not as popular as they used to be, they are still used in many field where they are unmatched in terms of performance.

IBM The current IBM's Operating System is `z/OS`, which is a *Monolithic* Operating System, it is being opened to different programming languages, such as `Java`, beforehand, however, it was only available in `COBOL`.

Batch Processing Mainframes are based on the *Batch Processing* paradigm, where a set of jobs are submitted to be executed in a *batch*, that is, a set of jobs that are executed in a single run, this is in contrast to *Real Time Processing*, where jobs are executed as soon as they are submitted.

This, provided that the company has a sufficient amount of jobs to be executed, allows for a continuous flow of work, and hence, a higher level of productivity.

2 Process Modeling

2.1 Modeling Business

In order to properly formalize the concepts that are going to be introduced through the rest of this course, we will introduce *mathematical models* that allow us to describe the ebbs and flows of human economy.

Processes These models are called *Processes*, there are certain common classes of projects, such as:

1. Service
2. Support
3. Management and Control
4. Physical
5. Information
6. Business

2.2 Process Descriptors

These processes can then be described by some diagrams:

- Hierarchical
- State Diagrams (Automata)
- DFD – Data Flow Diagram
- WIDE – Workflow on an Intelligent and Distributed database Environment
- Action Workflow
- Petri Nets

All of these give an intuitive way to describe the processes that are going on in a company, allowing for better understanding of the company's operations.

2.2.1 Hierarchical Process Model

Everything in a company can be described as a set of *Hierarchies*, that is, a tree of *Processes* that are organized in a *Top-Down* fashion, where the *Top* process is known as *Macroprocess*, the hierarchy goes as follows:

1. Macroprocess – Sales
2. Process – Sales Management
3. Phase – Order Processing
4. Activity – Shipment
5. Operation – Pricing, Packaging, etc. . .

Naturally, these hierarchy together form a *Forest.*

2.2.2 Data Flow Diagrams

Data Flow diagrams are what we also call *Flowcharts*, they are a graphical representation of the flow of the data through the company's Information Systems, they are not used often due to them being subject to *spaghettification*, that is, the diagrams can easily become too complex to be understood.

These flowcharts are usually composed of a set of components that are used to represent the different parts of the system, they are:

- Processes – Circles
- Data Collections – Rectangles
- Interface – Bordered Rectangles
- Data Flows – Directed Arrows

Definition 2.2.1 (Data Flow)**.** *A* Data Flow *represents any kind of flow in a system, the first component* must *be a process,the second can be either a process, a data collection or an interface, moreover, they can be either*

1. *Elementary*
2. *Structured*

Data Dictionary Usually, to help with readability, we provide a *Data Dictionary* and a textual description of each process, to help the user understand what the process does.

2.2.3 WIDE

WIDE Relies on three main components:

1. Process Model – Describes the activities in the system
2. Information Model – Describes the data being processed
3. Organization Model – Describes the structure and agents that are involved

Anti-Spaghetti Techniques The WIDE model introduces some more complex components with respect to the DFD model, such as forks and joins, to better describe the flow of the data through the system with fewer connections between the components.

Remark 2.2.1. *An analyst's role is to translate the business model in a way that is understandable in layman's terms, that is, when you are describing a process through a graph, strive to be clear, the important thing is that* People Are Going to Read It*.*

In conclusion, the WIDE model is just a more *expressive* version of the DFD model, which allows for more concise flowcharts.

2.2.4 Petri Nets

Petri Nets are a formal model that is used to describe the flow of data through an Information Systems

Definition 2.2.2 (Petri Nets)**.** *A petri net is a 3-Tuple $:= (P, T, A)$ that forms a* Bipartite Graph*:*

$$G(V, E) := \begin{cases} V & := (P \cup T) \\ A & := E \subseteq (P \times T) \cup (T \times P) \end{cases}$$

Where:

- *P is a set of places*
- *T is a set of transitions*
- *A is a set of arcs (Edges) that connect places and transitions in a* Directed *fashion*

Definition 2.2.3 (Marking)**.** *A* Marking *of a petri net is a function*

$$M : P \to \mathbb{N}_0.$$

Mapping a place to a non-negative integer, that is, the number of tokens in that place.

At each step of the evolution of the Petri Net, the marking function is updated to reflect the new state, for example, M_0 is the initial marking and M_f is the final marking.

Essentially, a Petri Net is a Finite State Automata in which a set of markings indicate the states that are 'Firable', once a transition happens, the markings are updated, and the process continues, so we are effectively running multiple Finite State Machines in parallel, each with the same topology.

Terminology Each transition $t \in T$ has an input set $^\circ t \subseteq P$ and an output set $t^\circ \subseteq P$ also called input and output places, the same notation also applies to places.

Petri Net Evolution An enabled transition can *Fire*, deleting a token in each input place and creating a token in each output place:

$$\begin{aligned} M_0 &= (2, 1, 0, 0, 1) \\ M_0 &\to M_1 \\ M_1 &= (1, 0, 1, 1, 1) \end{aligned}$$

A *Firable Sequence* is a sequence of transition $\sigma = \langle t_1, t_2, \ldots, t_n \rangle$ where

$$M_0 \to^{t_1} M_1 \to^{t_2} M_2 \ldots \to^{t_n} M_n.$$

For which we can use the closure notation $M_0 \to^{\sigma} M_n$ to denote the evolution of the markings.

Further Definitions Now, given a net

$$P = (P, T, A, M_0).$$

We have that:[1]

- A *Potentially Firable Transition* $t \in T$ is such that

$$\exists \tau \in T^{\star} s.t\ \tau t \text{ Is Firable}:\ {M_0}^{\tau} \to^{t}\ .$$

- A *Potentially Firable Sequence* $\sigma \in T^{\star}$ is such that there exists a prefix sequence $\tau \in T^{\star}$ such that $\tau\sigma$ is firable ($M_0 \to^{\tau\sigma} M_n$)

[1] $T^{\star}$ is the set of all sequences of transitions, so τt is a sequence of transitions that starts with τ and ends with t, hence we are using notation coming from regular expressions (Concatenation, Kleene star, etc. . .)

- A *Reachable Marking* M is such that
$$\exists \sigma \ s.t \ M_0 \to^{\sigma} M.$$
- $R(M_0)$ is the set of all reachable markings.
- P_r is the set of reachable places s.t
$$P_r = \{p \in P | \exists M \in R(M_0), M(p) > 0\}.$$

Petri Nets and Automata We have that Petri nets are a generalization of automata under particular conditions:

Definition 2.2.4 (Petri Nets and Automatas)**.** *A Finite state machine is a petri net where, for each transition t both the input and the output places have cardinality* 1*:*
$$\forall t \in T, |^{\circ}t| = |t^{\circ}| = 1.$$

2.2.5 Workflow Nets

Workflow nets are a generalization of Petri Nets, in which we have two additional conditions:

1. A source s.t $|^{\circ}t| = 0$ for all $t \in T$[2]
2. A sink s.t $|t^{\circ}| = 0$ for all $t \in T$

Hence for each node the node is *reachable* from the source and *can reach* the sink.

Workflow Nets as Models Workflow nets are more representative of real world systems, since they can model *producers* and *consumers* in a system, which are widespread in real world systems.

[2] AI generated math, please check (will fix once the slides are up).

3 Petri Nets – Cont.

3.1 Introduction

Petri nets are, as aforementionedly introduced, a *Graphical* and *Mathematical* modelling tool, that is, they are formalizable through math and representable pictorially.

Graphical Representation The graphical representation is a bipartite graph, where we have two kind of nodes:

- Places
- Transitions

The transitions are stylized as *black rectangles*, in order to distinguish them more easily, but they are easily interchangeable with the usual labeled circles that are used to represent states in a finite state machine.

Tokens are pictorially represented with dots, however, this is not that scalable, so we can also use mathematical notations and label a node with a number $n \in \mathbb{N}$ to represent n tokens in the place, be careful with this representation since it reduces the expressiveness of the model, especially towards non-mathematics oriented people.

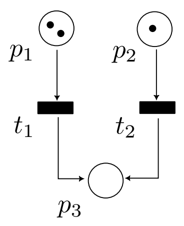

Figure 3.1: A Petri Net with 3 places and 2 transitions, we can see three tokens in p_1 and one token in p_2.

Matemathical Representation Since petri nets are just a graph, they can be rigorously formalized, as in definitions 2.2.2 and 2.2.3, so we omit the formalities here to avoid redundancy.

Transition Enabling We can further complicate a Petri Net by introducing the concept of transition *enabling*: a transition is enabled if it has enough tokens in its input places to fire, and it is disabled otherwise, naturally, the firing of the transition removes a token from each input place and adds a token to each output place.

The system steps in time *discretely*, that is each event happens simultaneously in a single step in time.

3.2 Petri Nets and Threads

In the previous chapter, we drew a parallel between Petri Nets and Finite State Machines, stating that their behaviour is *similar* to that of a set of FSM running in parallel, in fact, this parallel is not *entirely* made up, since we can notice that some common issues of parallelism such as conflicts are present in Petri Nets.

Definition 3.2.1 (Petri Net Run). *A run of a petri net is a finite or infinite sequence of markings and transitions:*

$$M_0 \to^{t_0} M_1 \to^{t_1} \dots \to^{t_{n-1}} M_n \to^{t_n} \dots .$$

such that M_0 is the initial marking, each transition t_i is enabled in the marking M_i and the marking M_{i+1} is the marking that results from firing t_i.

Execution Properties Now we can define some other concepts relative to the run of a petri net:

- Sequential Execution – Happens when transitions are on a *chain*, that is, there is a path from one transition to the other, imposing a *precedence constraint.*

- Synchronization – When multiple places are connected to a single transition, we have a *synchronization* between the places, that is, the transition will only fire when all the places have a token.

- Merging – When multiple transitions are connected to a single place, we have a *merging* of the transitions, that is, the place will only have a token when all the transitions have fired, reducing the number of total tokens.

- Forking – When a single transition is connected to multiple places, we have a *forking* of the transition, that is, the transition will fire multiple times, increasing the number of total tokens.

- Concurrency – When multiple transitions are connected to multiple places, we have *concurrency*, that is, the transitions can fire independently of each other, and the places can have tokens independently of each other.

3.2.1 Non-Determinism

Given the previously explored execution model of petri nets, one more thing must be said: their *evolution* (that is, the sequence of markings that they go through from their initial marking), is Non-Deterministic, hence, any enabled transition can fire at any time, and the marking can change at any time, this is a very important property of petri nets, since it allows us to model *concurrency* in a very natural way, but, as said before, it also opens the door to *conflicts*.

Definition 3.2.2 (Conflicts)**.** *A conflict is a pair of transitions t_i, t_j such that:*

1. *t_i and t_j are both enabled in the same marking M.*
2. *t_i and t_j are connected by a path in the graph.*
3. *The firing of t_i leads to the disabling of t_j and vice-versa.*

These conflicts are not too dissimilar to what was explored in the Operating Systems course with the *deadlock* concept[1], they can be resolved by introducing *priority* to the transitions, that is, we can restructure the mathematical definition of a Petri Net to allow for a hierarchy of transitions, or we can simply resolve the conflict by changing the network topology.

[1] Specifically the dining philosophers problem.

3.3 Reachability Analysis

A marking is said to be *reachable* from another marking if there exists a sequence of transitions $\langle t_1, t_2, \ldots, t_n \rangle$ that when fired, lead from the first marking to the second.

Definition 3.3.1 (Reachability Set)**.** *The* reachability set *of a Petri Net is the set of all the markings that are reachable from the initial marking, one can think of this as a form of* closure*.*

Given the topology of a Petri Net (The set of its places and transitions), we can then uniquely determine the state the network is in by its markings, each of these being a unique element of the reachability set.

Since our petri nets describe a real-world system, it might be helpful to know if, from our current state, we are likely to reach a particular outcome, the process that determines this is called *reachability analysis*, in which we try to solve a membership problem given a marking and the reachability set of the Petri Net.

3.3.1 Reachability graphs

Reachability graphs are a way to represent the evolution of a Petri Net, they are a directed graph, where each node is a marking of the Petri Net, and each edge represents a transition that can be fired from the current marking to the next marking.

Reachability Graph Construction Reachability graphs are constructed by starting from the initial marking, and then, for each marking, we generate all the possible markings that can be reached from the current marking by firing a transition.

By properly identifying the frontier nodes, the generation pf the reachability graph is performable in a finite amount of steps, even if the Petri Net is unbounded, this is polynomial time, and therefore the reachability graph can be generated in a finite amount of time.

In general, there are three types of frontier nodes:

1. Terminal – no possible transition
2. Duplicate – already generated
3. *Infinitely Reproducible* – can be generated infinitely many times

Infinite Reproducibility A marking $M'' \geq M'$ is *Infinitely Reproducible if*:

$$M'' \geq M' \tag{3.1}$$

$$m''_i \geq m'_i \quad \forall i \in (1, 2, \ldots, n) \tag{3.2}$$

If the condition holds there exists a sequence that leads from M' to M'', and that is surely firable infinitely many times, hence, the marking is infinitely reproducible, in that case, the Petri Net is said to be *unbounded.*

For an unbounded Petri Net, we can indicate an arbitrarily large amount of tokens through the symbol ω

3.4 Petri Nets Extensions

Petri nets can be *extended* in order to model more complex systems, for example, we can have:

- Arc Multiplicity
- Inhibitor Arcs
- Priority Levels
- Enabling Functions (Guard Conditions)

Remark 3.4.1. *The last three extensions* destroy *the infinitely reproducibility property of the Petri Net.*

3.4.1 Arc Multiplicity

An *arc cardinality* may be associated with input and output arcs, whereby the enabling and firing rules change:

- Each input must contain at least as many tokens as the input arc's cardinality
- When the transition fires, the number of tokens removed from each input place is equal to the input arc's cardinality, as is the number of token added to each output place.

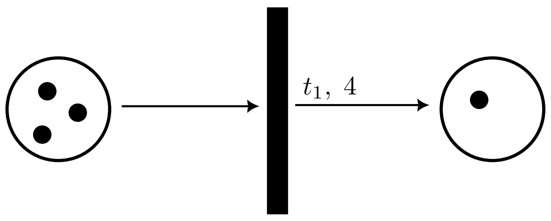

Figure 3.2: A Petri net with arc multiplicity, note that t_1 has *no way* to fire since it requires 4 tokens in p_1 but it only ever has 3.

3.4.2 Inhibitor Arcs

Inhibitor arcs are arcs represented with a circle head, the transition can fire only if the inhibitor place does *not* contain any tokens.

They are used to implement constraints on the firing of a transition, in the restaurant example, we could think of a transition that represents the *delivery* of a dish, and we could have an inhibitor arc from the *delivery* transition to the *payment* transition, so that the dish can only be delivered once the payment has been made.

Inhibitor Arcs and Multithreading Once again, we also notice a stunning similarity with multithreading, where we can think of the inhibitor as some sort of synchronization primitive, such as a *mutex.*or a *semaphore.*

Inhibitors and Transitions An inhibitor arc placed on a transition means that the transition can only fire if the inhibitor place does *not* contain any tokens, however, the cardinality of the output arc is not affected by the presence of the inhibitor arc, nor is the inhibitor affected by the cardinality.

3.4.3 Priority Levels

A Priority level can be attached to each Petri Net Transition, the standard execution rules are then modified to allow the highest priority to fire first in case of conflict, when two transitions have the same priority, that subset of transitions fires simultaneously.

3.4.4 Enabling Functions

An enabling function (or guard) is a boolean expression, composed with Petri Nets primitives, the enabling rules are modified so that, besides the standard conditions, the transition can only fire if the guard evaluates to *true.*

Hence we can formally define an enabling functions as:

$$\begin{aligned} f(tk) &:= (P, T, A) \cup (+, \cdot, \leq, \geq, \neq, \\ &\quad \wedge, \vee, \neg, \text{true}, \text{false}) \to \{\text{true}, \text{false}\} \end{aligned}$$

But, in practice, we often adopt a verbal description of the boolean predicate that the enabling function implements[2]

[2] Remember, *someone is gonna read this*, and it's *not* going to be a mathematician.

3.4.5 Colored Petri Nets

We can extend petri nets with colors, to increase expressiveness:

- We say that C is a set of colors of cardinality $|C|$ and x is a color of C.
- Place p can contain tokens of any color $x \in C$
- Transition t can fire tokens of any color $x \in C$

3.4.6 Stochastic Petri Nets

Petri nets can be extended by clearly associating *time* with the firing of transitions, resulting in *Timed Petri Nets*.

A special case of timed Petri Nets is stochastic petri nets (SPN), where the firing times are considered to be random variables.

This can be useful to model the random natures of sales, that are based on a number of difficuly-to-predict factors, such as the customer's tastes, current season, advertisements, world events, etc…

4 Seminar

4.1 Background

The seminar in question is a mandatory seminar given on the 20th of march 2023, it is about *business transformation*, specifically, we are interested in the story of a company[1] that traded in the supply and distribution of scientific publications.

The Business Model The company's service was regulated by tenders and contracts and provided a high profit margin, based on supply fees, since back in the day intra-national trade was hard, the service included the identification of the best suppliers, and of the products itself.

Essentially, the company's role was to, given the customer's need of an academic publication/book, to find the best supplier, and to organize the shipping of the product to the customer.

4.1.1 Business Transformation

The company decided to engage in a $SWOT$[2] analysis to transform its business model, we are replicating this in the classroom:

- **Strengths** – High profits, captive market, limited competition
- **Weaknesses** – Complicated business model, limited demand and small market size
- **Opportunities** – Market expansion towards EU, new technologies, new product categories (not only scientific publications)
- **Threats** – University budget cuts, competitors on delivery and price

From this, it is clear that the risks and the weaknesses are threatening the company's business model, and that the company's strengths might be at risk, from there, it was clear that these opportunities must be exploited.

[1] It is a personal belief of a few of us that the company in question might be Springer, but we are not sure.
[2] Strengths, Weaknesses, Opportunities, Threats

No Plan For this company, there has been *no* formal transformation plan, the transition occurred organically, with resources being allocated on the go

4.2 Digression – Strategic Plan Design

A strategic plan is a document that describes the company's goals, and the way to achieve them, it is a *roadmap* that the company follows to achieve its goals.

it is split in 4 phases:

1. **Needs** – What are the needs of the company? a strategic plan (such as one necessary for this transition), starts with the identification of the needs of the company, such as which opportunities to take and which strenghts to preserve
2. **Drivers** – What are the forces that can be applied that will help the company to achieve its goals?
3. **Objective Key Results** – Given the needs and the drivers, we can now define the objectives, and the key results that will help us to achieve them.

   We gather a list of goals to accomplish and we utilize some metrics such as KPIs to track their progress.
4. **Budgeting and Planning** – After the roadmap is complete, we have to reason on our *ability* to reach the goals, and we have to allocate resources accordingly, both in terms of monetary costs and in terms of time.

4.3 The Transformation

Our company decided to transform its business model, and to do so, it decided to pursue the following opportunities

- Expand their catalog to include consumer products (such as books, movies, etc. . .)
- Reach consumers directly, instead of relying on universities
- Reach the EU market

We can then analyze this thrown-together plan through the lens of our 4-phase plan.

Needs As stated before, the company's needs are mainly relative to *expansion*, both expansion of their offering and expansion of their market, they also needed to better define their offering through electronic catalogs.

Another one of the company's needs was to actually let the customers know that this new offering was available, and that they could buy it directly from the company.

The proposed **solution** was to exploit *e-commerce* to solve both needs at once, to both reach international markets, act as a catalog and a bridge to new potential customers.

Results The company's results were not as expected, the transformation resulted in:

- Little increase of orders
- *High* ratio of supply-chain failures
- Consequentially, customers' dissatisfaction

This stemmed from the fact that the supply chain advertised products that *weren't there*, their availability was not guaranteed, and depended on the publisher and the supplier. This was *not* advertised to the customer, and therefore led to an overload of orders that the company could not handle.

4.3.1 Fixing the mess

From this, a new set of needs arose:

- Reduce customer frustration
- Define what's *really* available
- Reduce wasted time

This was solved by integrating the supply chain with the e-commerce platform, by asking the suppliers the status of the product on a daily basis, and by buffering the products in a warehouse, the company implemented a real-time inventory system, and was able to guarantee the availability of the products directly from their website.

Unfortunately, this was not enough, the company's *business model* was still flawed, and the company was still losing money, little increase of orders was observed, even though the failure-to-supply events were reduced.

Another try Next, they identified another set of weaknesses that, if solved, would allow them possibly reduce customer frustration and increase the number of orders, those were:

- Reduce delivery time
- Reduce transportation costs

Further integration with the supply chain was implemented, with the construction of databases and APIs that allowed the company to track the status of production and shipping of the products, and to track the status of the products in the warehouse.

Finally, Results This integration allowed the company to reduce the supply times and, most importantly, to reduce the costs, increasing its profit margin, Unfortunately, this was not enough, as during this time, the company's main competitor, Amazon[3], started to offer the same service, and the company's market share started to decrease, the company was now forced to provide a service that was at least as good as Amazon's, and to do so, it had to reduce its prices.

4.4 Amazon

The arrival of Amazon forced to somewhat emulate their business model to improve their competitiveness, and to do so, they had to integrate their platform with marketplaces, funnily, the company achieved this by integrating with Amazon's own marketplace, using it as a platform for distribution of their products, during this timeframe, the increase of orders was significant, and the sales from amazon's marketplace overtook the sales from the company's own website.

4.4.1 New Opportunities

The arrival of this new framework allowed the company to identify new opportunities, such as *Triangulation* of import-export flows.

Definition 4.4.1 (Triangulation)**.** *Triangulation is the act of intercepting a product in one country, and then shipping it to another country, in order to make a profit from the difference in price between the two countries.*

This is often possible due to the fact that some products might *not be available in some countries, and therefore their price is higher than in other countries, the same can happen if the product is available, but the price is higher due to differences in the cost of living/pro-capita income.*

[3] Cue final boss music.

Long Tail Another opportunity is specializing on the *Long Tail* of the market, which is the set of products that are not frequently sold, and are therefore outliers in the distribution of the products sold[4], these are probably products where the economy of scale of services such as Amazon is not applicable, and therefore our company can establish a competitive advantage by specializing on these products.

4.5 The End

Did this transformation work? the answer is *no*, the strong competition still forced the company to reduce its margins, while its dependence on marketplaces as a distribution channel made it vulnerable to outside influences from the marketplaces themselves, which, coincidentially, were also the company's main competitors.

Failure Ultimatedly, the company failed due to insufficient business margins, hence there is no definitive moral to this story, except that *business is hard.*

[4] which is gamma shaped in most cases, hence the *long tail* metaphor.

5 Software Size Estimation

5.1 SLOC

In order to decide whether engaging in a refactoring/rewrite of a software-based project, it is important to estimate the size of the project, both of the existing codebase and of the hypothetical new codebase, this is important since it seems to correlate with the difficulty and the cost of the project.[1]

A metric for size The most common [2] metric is *SLOC*, or *Source Lines of Code*, which is the number of lines of code in the project, this is easy to calculate mathematically, and can even be made more specific by discounting comments and blank lines, or, by only counting lines in core functions and hot paths.

This method is terrifyingly inaccurate as it suffers from a number of flaws

- Different languages have different line lengths, and therefore different SLOC counts
- Inefficient library function calls take up less lines of code than efficient, inbuilt ones
- 80% of code is used 20% of the time, and vice-versa[3], hence SLOC is not a good metric for estimating the size of the project since it poorly correlates with the actual complexity of the project.

There are some magic numbers proposed to calculate a SLOC index by extracting particular features from the codebase, such as the number of used files, functions, lines etc..., and then feeding this to a linear regressor, such as:

$$SLOC(X|\theta) := \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{n-1} \\ \theta_n \end{bmatrix} \begin{bmatrix} 1 \\ X_1 \\ \vdots \\ X_{n-1} \\ X_n \end{bmatrix}.$$

[1] At least that is what they thought in the 90s. [2] and useless. [3] Like the 80/20 rule, but for code.

This is as scientifically accurate as predicting the outcome of battles from the flight patterns of birds, but it has the advantage of using math symbols, therefore **Science!**

5.2 FP

Function point models aim to estimate the size of software in terms of user functionalities (use cases exposed by the frontend) Function points are not only a predictor of size, but also a better predictor of difficulty, since they more closely measure the used parts of the project, and therefore the complexity of the project.

Unfortunately, this is still presented as a *magic number* that is calculated by a linear regressor, such as

$$BFP := 4 \cdot EI + 5 \cdot EO + 4 \cdot EQ + 10 \cdot ILF + 7 \cdot EIF.$$

Where:

$$\begin{aligned} EI &:= \text{External Inputs} \\ EO &:= \text{External Outputs} \\ EQ &:= \text{External Inquiries} \\ ILF &:= \text{Internal Logical Files (Databases and directories)} \\ EIF &:= \text{External Interface Files (Libraries)} \end{aligned}$$

With a $\pm 25\%$ error margin, given by complexity.[4]

Variants Naturally, there are a number of variants of this model, of course, creating a new variant is quite easy, all that is needed is coming up with a new set of *magic numbers* to parametrize the regressor, you can come up with such thing yourself!

FP were initially devised for business application, but there has been a push on trying to adapt

[4] According to this model, importing a Linear Algebra library (such as `GLM` or `numpy`), is more complex than implementing a neural network from scratch, or writing your own matrix library.

them to other domains such as scientific or real-time applications.

FP to SLOC conversion Sometimes it is necessary to convert FP to SLOC, we do this through *further magic numbers*:

Language	SLOC/FP 1	FLOC/FP 2
Assembler		

Table 5.1: Conversion Tables

5.3 Object Points

OP is a metric that tries to provide an equivalent metric to Function Point in an object oriented context, it utilizes object counts instead of function counts, and therefore is more accurate in object oriented projects, where classes are omnipresent.

Objects are **not** raw object classes, rather, they are **abstractions** of the objects, and are to be counted whenever an entity that resembles a class is encountered in the codebase.

OP to SLOC conversion There is *no* established standard for OP, and therefore conversion from OP to SLOC is also non-standardized

5.4 Wideband-Delphi

This model can be applied at the beginning of a project, and then updated as the project progresses, Usually it is used to examine a small section or component of the overall, larger project.

Steps:

1. A group of experts is each given the program's specification

2.

6 Project Management

6.1 Context

Project management is **key** to a project's success, first, to talk about project management, we define a *project*:

Definition 6.1.1 (Project)**.** *A Project is defined as a group of* stakeholders*, usually under some measure of coordination or management, that pool together resources to achieve a goal.*

The act of project management is necessary to ensure that schedules are respected and that resources are utilized efficiently, failures in project management almost always resolves in a loss of project, hence proper management is fundamental.

6.2 Project Management Models

In the previous chapter, we observed dozens of models for code complexity estimation, in this case, there is a *single* established method for project modelling, it is called PERT and is based on *Petri Nets*, which we already introduced in chapter 3.

Terminology We can introduce some terminology used in PERT petri nets:

- Merge Events – Merge events are represented by a place collecting multiple incoming transitions, hence they act as a *barrier*, synchronizing the separate sub-activities
- Burst Events – Burst events are represented by a place triggering multiple outgoing transitions, hence they act as a *fork*, launching multiple sub-activities
- Dummy Activity – Dummy activities are used to represent a *connection* between events, however, since they not a representation of a real activity but rather a *control flow* object, they do not consume any resource nor do they spend any time to be completed; they are represented by a dotted arrow.

Errors to be avoided In the construction of a project workflow petri net, some more constraints should be considered, in addition to those of regular petri nets:

- Multiple identical transitions – there can't be two transitions with the same start and end place, hence, one needs to use dummy variables to model this in order to be mathematically rigorous.
- Loops – looping should not be performed, even if allowed by the mathematical model, since the project *must* have a beginning and an end.
- Multiple Start Events – The petri net should be a workflow net, hence, it should have a single start and end state
- Correct Direction – The network should flow from left to right (or right to left depending on dominant cultural bias in your organization[1])

Critical Path Analysis The critical path for any network is defined as the *longest path* in any network.

[1] We want to be as PC as possible.

7 Business intelligence and Data Warehousing

7.1 Data Warehousing

Data warehousing is a *concept*, it is, formally

Definition 7.1.1 (Data Warehouse)**.** *A collection of methods, technologies and tools to assist the* knowledge worker[a] *to conduct data analysis aimed at supporting decision-making and/or improving the management of information assets.*

[a] for example a manager or an analyst

The data collected in a data warehouse is:

- Integrated – from multiple sources
- Consistent – despite its different origins
- Focused – with a specific interest area
- Historical – over a consistent timeframe
- Permanent – it is not a temporary storage

Purpose A data warehouse **helps** in:

- Taking decisions
- Identifying phenomena
- Forecasting the future
- Controlling a complex system

Remark 7.1.1. *The data warehouse does* ***not*** *do any of the above, it merely* ***helps*** *by providing intelligence (i.e information) to the knowledge worker.*

7.1.1 OLTP & OLAP

The two main data analysis techniques are OLTP and OLAP, respectively:

- OLTP – Online Transaction Processing
  - Write/Read transactions
  - Consistent
  - Many, fast and frequent transactions
  - Highly concurrent
  - Accesses a small subset of the data
  - On-the-fly data update
- OLAP – Online Analytical Processing
  - Read-only
  - Few operations
  - Low concurrency level
  - Accesses huge amounts of data
  - Data is historical and mostly static

Each is commonly associated, respectively, with an *Operational Database* and a *Data Warehouse.*

7.1.2 Operational Databases vs Data Warehouses

Operational databases are designed to support OLTP, whereas data warehouses are designed to support OLAP, we can highlight the following differences:

- Different computational loads – OLTP is based on concurrency and responsiveness, whereas OLAP is usually based on batch processing
- Different needs
  - DB – Dynamic Data, asynchronous updates
  - DW – Static Data, batch updates
- Different integration
  - DB – Suppporting operations (focused, timely, responsive)
  - DW – Supporting decisions (descriptive, historical, meticulous)
- Different data collection
  - DB – minimal, just what is needed to support short-term operations

  - DW – maximal, all data that is relevant to the organization, to be used for long-term analysis

Moreover, the two models also have a different approach to common Relational Databases concepts, such as:

- Redundancy
  - DB – Avoided, leads to inefficient updates
  - DW – Accepted, leads to efficient queries
- Indexing
  - DB – Good for reads, bad for writes, hence, the level of indexing must be carefully chosen to balance the two
  - DW – Good for reads, there are **no** writes, hence, the more the better

Common Systems The usual industry players are also present in this field, companies such as IBM, Microsoft and etc… provide their own data warehouse solutions.

7.2 Data Warehouse Architectures

In general, the issues that are addressed by any software engineer are also present in the data warehouse field, things such as *scalability*, *security*, *extensibility*; we are also interested in *two* extra qualities:

1. OLTP and OLAP separation – The two systems, as explored before, are remarkably distinct in terms of their requiremets, one being read only and the other requiring a high level of responsiveness, hence, by separating them, we can target them with more focussed optimizations, thus, improving performance.

   Also, by separating them, we can have a more *modular* system, which works towards our other goals (scalability, extensibility, etc…), and, finally, we can also devise different storage solutions for each, which can be, for example, a *relational database* for the OLTP and a *data warehouse* for the OLAP.

2. Administrability – The system should be easy to administer, this means that a single person should be able to manage the whole system, without having to know the details of each component.

Choosing the architecture Naturally, the architecture is dependent on *design choices* that are guided by the *business requirements* and the *technical requirements.*

These then determine the costs of operation, the utilized resources (such as the software system), the level of possible integration with other systems, and, most importantly, the **cost** of data processsing.

7.2.1 Data Marts

A data mart acts as a *frontend* to a data warehouse, it is a *subset* of the data warehouse, and it is designed to support a specific *business area*, hence, it is used to provide intelligence to a specific *knowledge worker* through a set of APIs.

Data marts are **not** always subsets of the data warehouse, specifically, we have two kinds of data marts:

1. Dependent data marts – These are subsets of the data warehouse, probably an aggregation of data in the primary data warehouse.

2. Independent data marts – These are subsets of the **Operational Database**, they are usually used to provide *real-time* information to the knowledge worker.

Importance in the architecture Since data marts act as a *frontend* to the data warehouse, they are important in the sense that they represent the layer of abstraction that is actually used by the knowledge worker, hence, they are responsible with the knowledge worker's perception of the data warehouse.

7.2.2 Different Level Architecture

Depending on the number of layers of the architecture in which data is stored, we have a n-level architecture, where n is the number of aforementioned layers.

Level 1 Architecture This is the simples architecture, it consists of a single layer, which is the data warehouse, we have:

- A virtual DB, with no OLTP-OLAP separation

- Historic data coincides with current data (Operational DB + Data Warehouse)

- Difficult to integrate with other sources

Level 2 Architecture Moving on to a more complex architecture, we have a two-level architecture, where we complement the data warehouse with data marts and we accept different sources, we have:

- An Operational DB, coupled with other external databases
- An ETL layer – Extraction, Transformation, Loading; acts as a filter
- A Data Warehouse, with multiple data marts

We can even merge the concept of data marts with the concept of data warehouses, by feeding the data *directly* into data marts.

Level 3 Architecture Finally, we have a three-level architecture, where we have a data warehouse, An operational database, and a separation of the ETL activities into *two* phases, these are:

1. Extraction/Transformation
2. Loading

These feed into a *reconciled data* storage, which then feeds into the data warehouse.

7.2.3 Data Cleaning

The process of *reconciling* data goes through multiple stages:

- Extraction – gathers quantitative features from information
- Cleaning – modifies values, drops duplicates, detects inconsistencies and null values, spelling mistakes, abbreviations, etc...
- Transformation – transform formats (US dates to EU/ISO) to a common standard
- Loading – updates data, either through *refreshing*, that is, reloading the whole DB *ex novo*, or through differential updates

Metadata As an appendix, we also spend a few words on *metadata*, that is, data about other data, we have:

- Internal metadata – concerning the data warehouse's administration, for example, sources, transformations, schemas, users, etc...
- External metadata – which might concern users, such as measurement units utilized, possible data combinations, etc...
- Standards Utilized
- CWM – Common Warehouse Model, defined by:
  - UML
  - XML
  - XMI

7.3 Data Warehouse Multidimensionality

Data is *extracted* from the world following a precise sequence of events:

1. An event occurs, for example, Giovanni, age 26 buys a pizza
2. A datum is extracted by discarding irrelevant information from the event (`pizza`, `5$`, `April 17, 2023`)
3. A fact is obtained by aggregating data, as a measurements in N-dimensional space (sold N units of M articles on date D)

Data as tensors Hence data can be visualized as *tensors*, a concept that is quite common to us AI scientists.

This allows us to perform projections and aggregations on the data along certain dimensions, for example, we can aggregate the data along the `time` dimension, obtaining an average over a certain period of time, or we can project the data along the `product` dimension, obtaining the total amount of products sold.

Dimensional Hierarchy We can define a dimensional hierarchy, where, for each dimension, we have a hierarchy that groups together values at different levels of aggregation, for example, we can have a `time` dimension, where we can organize different units of time, such as *months*, *days*, *hours*, *minutes*, *etc...* as shown in fig. 7.1.

7.4 Data Warehouse Access

Access to a data warehouse is done through multiple means, these depend on the needs of the person accessing the warehouse, in fact, we can identify 3 different reasons for accessing the warehouse:

- Reporting – The user wants to extract a well-structured set of data from the warehouse, in order to present it to a *knowledge worker*, or at a meeting, this is easy to standardize and to automate, since the necessary data is known well in advance.

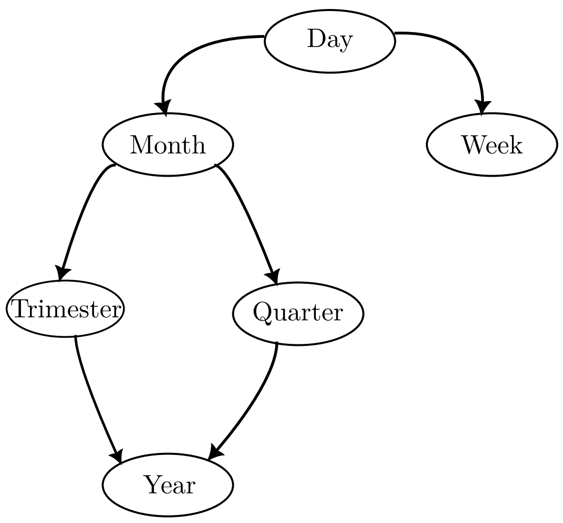

Figure 7.1: Here we have a dimensional hierarchy for the *time* dimension, the directed arrows denote functional dependencies, that is, the *parent* dimension is *higher* in the hierarchy than the *child* dimension, meaning that it has more *granularity*.

- OLAP – The user wants to perform *ad-hoc* queries on the data warehouse, in order to extract information that is not known in advance, this is more difficult and is usually done interactively (and manually).

- Data Mining – The user wants to extract data from the warehouse to satisfy needs that are not known in advance, similarly to OLAP, however, this is usually done in an almost *automatic* way, by means of *data mining* algorithms.

Let us remark the differences between plain reporting and OLAP:

Reporting Reports are so structured that obtaining them is a *routine* task, there can be queries to obtain reports, along with an array of automated tools to produce them.

Reports are usually organized in *presentations*, which can be easily distributed and comprehended by non-technical users.

OLAP OLAP is usually done in an *analysis session*, which can be divided into a series of steps, each dependent on the result obtained previously, this is akin to the use of a *jupyter notebook* to construct a *data science* pipeline. Typical users are domain experts, which are *not* necessarily computer scientists.

7.4.1 OLAP Operations

There are a number of operations that can be performed on a data warehouse, these are much akin the operations performed on a *relational database*, however, they are more *complex* and *powerful*, we will now list a number of them and discuss them in depth.

Restriction Restriction is the filtering of values based on a certain condition evaluated on a dimensional attribute, in case of equality with a single value, this is called **slicing**, since we are *slicing* the data along a certain dimension, making it go away.

Aggregation Aggregation is the application of a function to a set of values, such that this function reduces the set in dimensionality.

Distributive	Algebraic	Holistic
`SUM`	`AVG`	`COUNT`
`MAX`	`STDDEV`	
`MIN`	`MEDIAN`	

Table 7.1: Some of the most common aggregation functions.

Drill-Down It is expected that the deaggregated data is originally found in the data warehouse, since most aggregate operations are non-invertible.

The process of inverting this aggregation is called *drill-down*.

Slice-And-Dice Operations Keeping with our multi-dimension point of view, we can *slice* our data cuboid along a dimension in order to perform a *selection* operation, this is akin to the SQL operators `WHERE` and `HAVING`.

Pivoting Pivoting works as a *transposition* of the data, this is natural if we keep thinking of our data as being in the form of *tensors*, and can be done along any dimension.

Pivoting is common when visualizing the data in a *table*, since it allows us to *transpose* the data, making it easier to read, seeking such an example on the internet is advised in order to understand the concept.

Drill-Through Drilling **through** is a more extreme version of *drill-down*, where we access the data at a lower level of granularity, typically this is done by accessing reconciled data or operational databases directly, bypassing the data warehouse.

Drill-Across Drilling **across** is the process of correlating data from different dimensions, this is done by *joining* the data from different datasets, like SQL's `JOIN`.