AI and QFD share a common ancestry: the Perceptron, a collection of Ichikawa fishbone diagrams that somehow describe dependency. The perceptron was a very simple model of how neural networks think.

The original perceptron proved to be much too small to compete with natural thinking, but good enough to study cause-effect relationships between customer needs and technical characteristics.

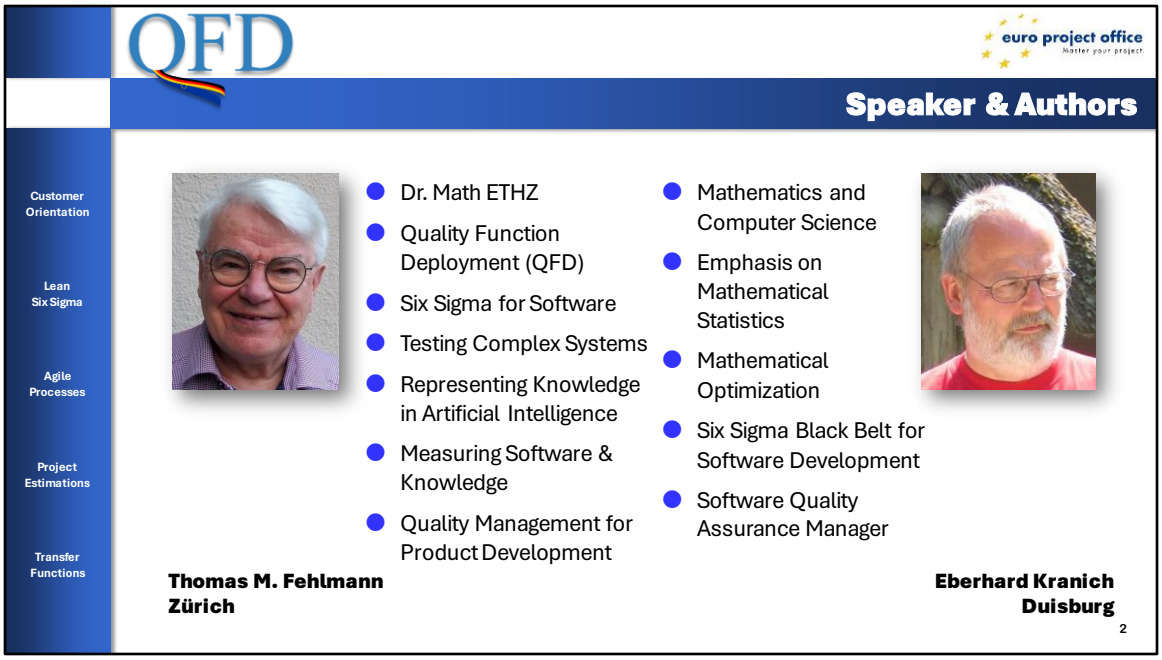
In contrast to the perceptron, QFD became a mighty tool in the Eighties. It allowed expert teams that knew the current domain to identify the causes that make a product successful in the market and to target products predictably to cover customer needs.

But in the 21st century, preferences have changed and people care more about social standing and reputation than about actual benefit. Collecting big data that can be processed by even bigger perceptrons – called Large Language Models (LLM) – replace at least partially the former QFD cause/effect analysis. With enough data, it seems that the only remaining relevant cause is adoption by enough people.

However, this works only partially. Expectations for future profits only partially cover today's real cost, and the famous AI that should process these big data collections unfortunately hallucinate. This happens because AI does not know anything about the current domain.

Thus, the question arises whether we cannot take the best of both sides and complement AI by sound cause/effect analysis with QFD.

This Café is about successful cases but also about strong barriers that make such common work difficult. Nevertheless, we also give indications on how to overcome such barriers.



Thomas Fehlmann, Zurich

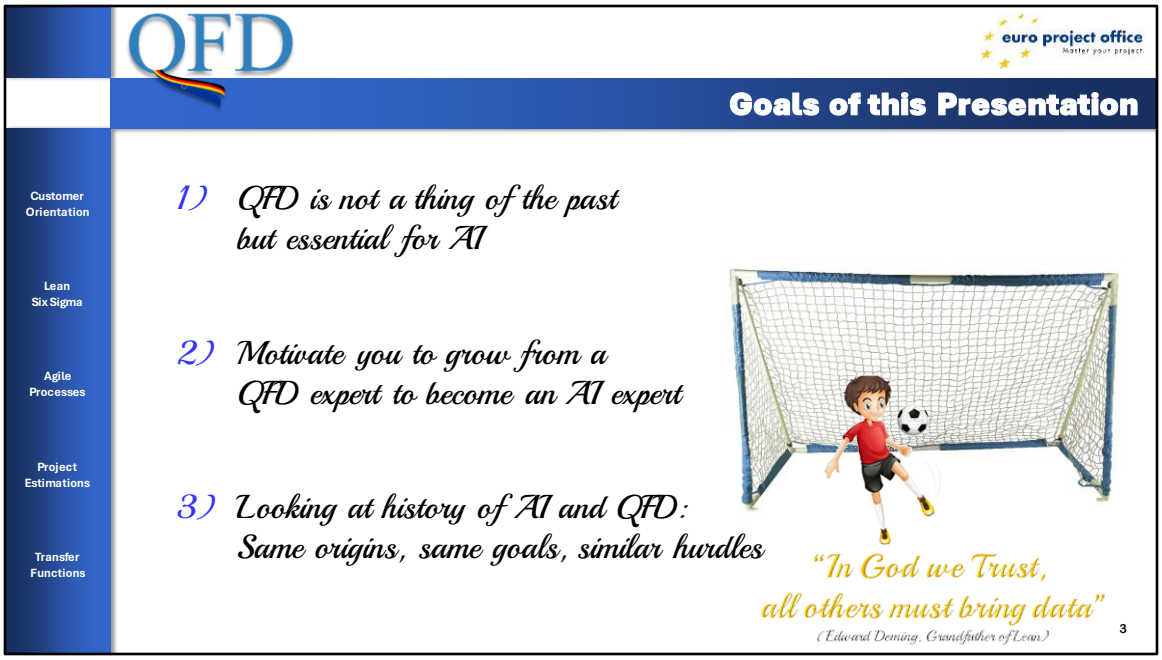
Dr. Thomas Fehlmann is an expert in software metrics and testing, a Lean Six Sigma Black Belt for agile software development and a promoter of customer-centric product design and testing. As a quality manager for software companies, he has led several companies to global market leadership using Quality Function Deployment (QFD) and Six Sigma for software.

He is Academic Member of the Athens Institute for Education and Research.

Eberhard Kranich, Duisburg

Eberhard Kranich studied Mathematics and Computer Science, with an emphasis on Mathematical Statistics, Mathematical Optimization, and on Theory of Polynomial Complexity of Algorithms.

He worked at T-Systems International GmbH in Bonn, Germany until 2013, as a Six Sigma Black Belt and Quality Assurance Manager, mainly in the context of software development.

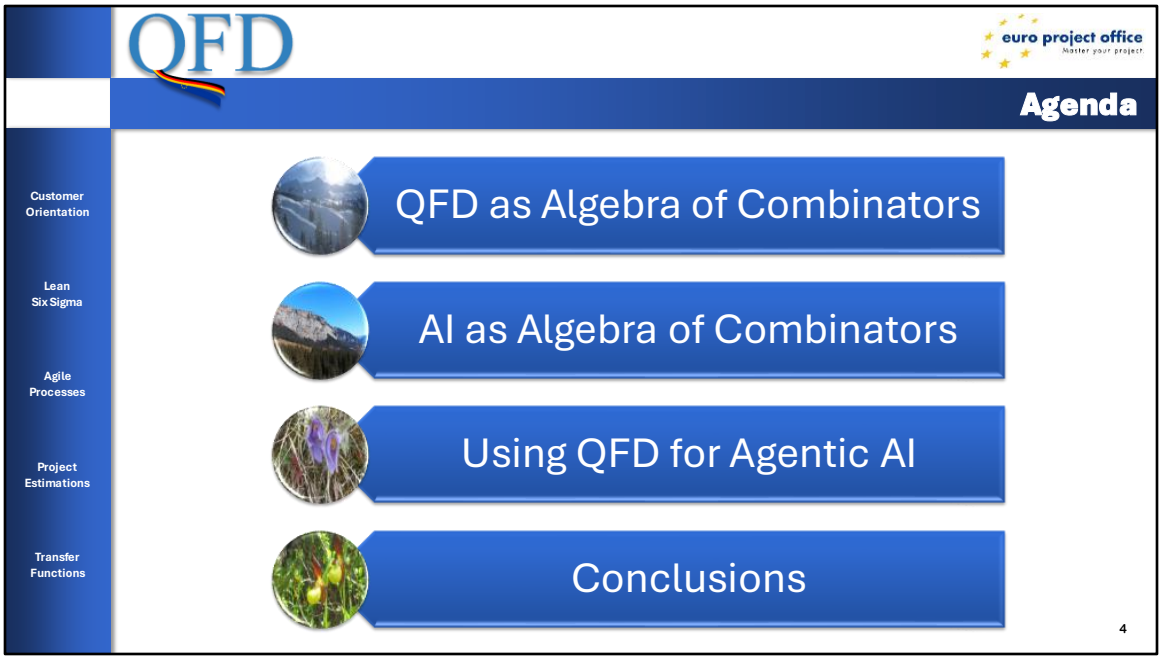


The focus on ISO 19761 COSMIC is due to the popularity of COSMIC when sizing embedded software. Any other sizing method can be adapted as well, if allowing for a little imprecision for not being compliant to the VIM and the GUM.

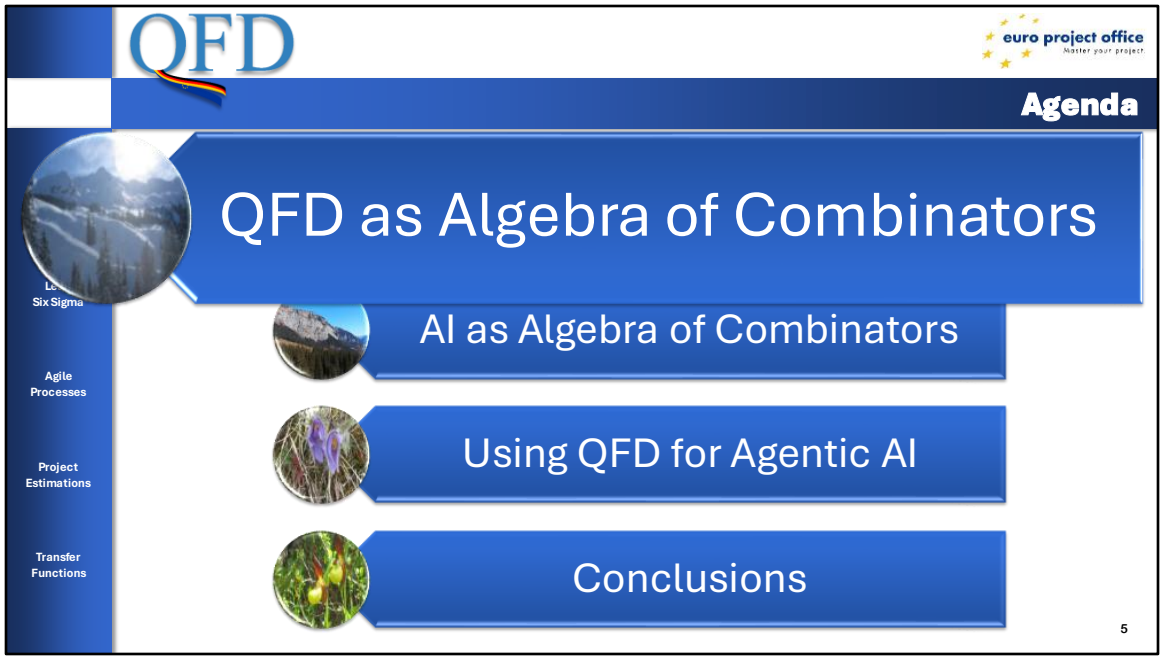
The VIM: ISO/IEC Guide 99:2007 International Vocabulary of Metrology – Basic and general concepts and associated terms (Vocabulaire International de Métrologie – VIM)

The GUM: ISO/IEC CD Guide 98-3, 2015. Evaluation of measurement data – Part 3: Guide to Uncertainty in Measurement (GUM).

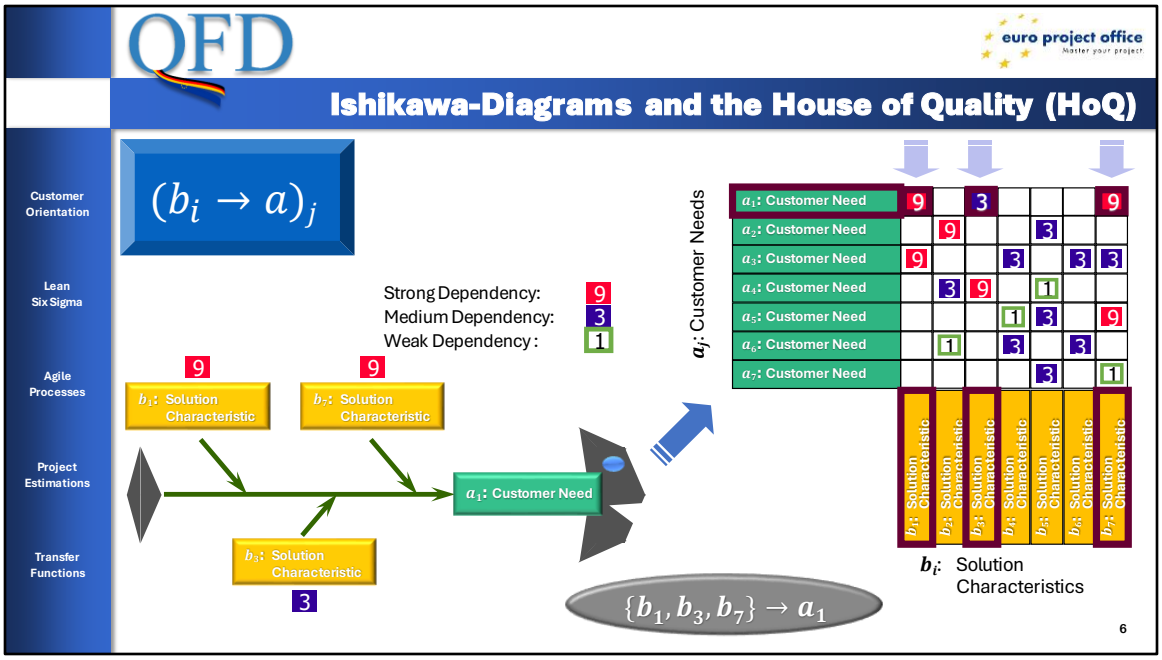
The VIM and the GUM are among the oldest standards that the world uses. In simple words: You can add & subtract measurements, like you can measure distance between several points in space, and get the total distance, possibly after some trigonometric adjustments.



- QFD as Algebra of Combinators
- AI as Algebra of Combinators
- Using QFD for Agentic AI
- Conclusions



- QFD as Algebra of Combinators
- AI as Algebra of Combinators
- Using QFD for Agentic AI
- Conclusions



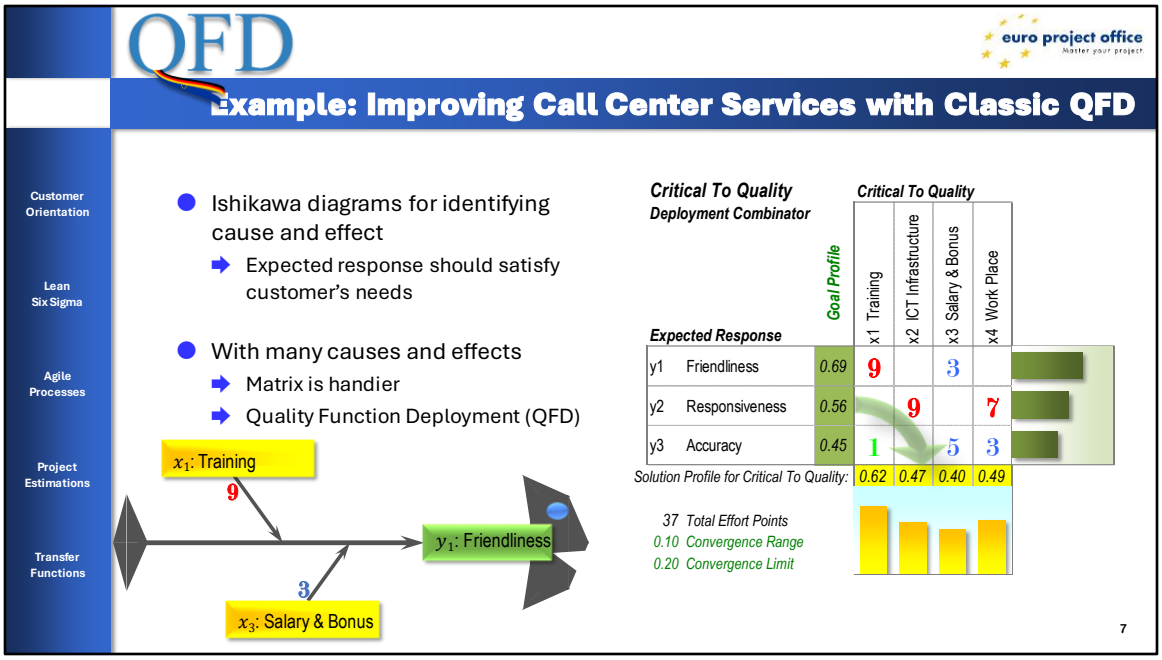
In 2001, Fehlmann presented a paper at the 7th International Symposium for Quality Function Deployment on "QFD as Algebra of Combinators" that was widely not understood. Its content said that QFD constitutes a model for combinatory logic; however, who knew combinatory logic? Combinatory logic is a foundation of computer science, explaining how programs rely on a quantifier-free logical structure that is Turing-complete but has the charming aspect that it looks so complicated that it only appeals to mathematical nerds.

Nevertheless, it was a simple observation looking at the origins of QFD as a matrix-representation of a series of Ishikawa-, or Fishbone-Diagrams. Such diagrams are of the form

$$b_i \rightarrow a$$

Here, b_i is a finite set of causes; a is the observable response.

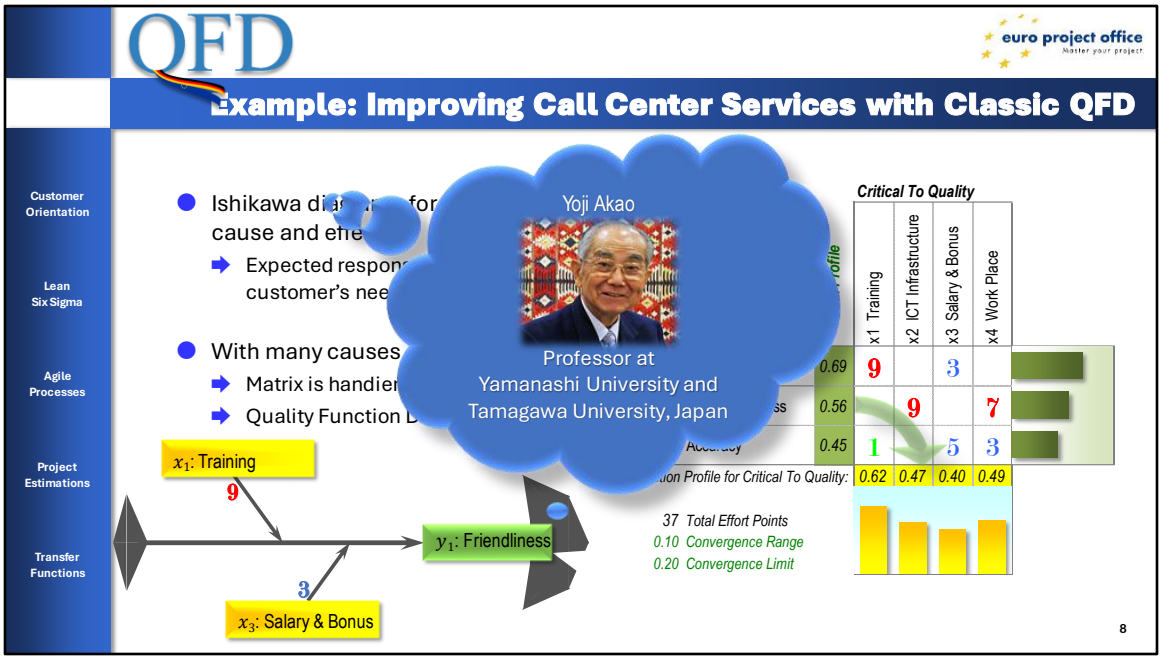
Moreover, a set of Ishikawa-Diagrams that refer to the same topics can be written as a matrix.



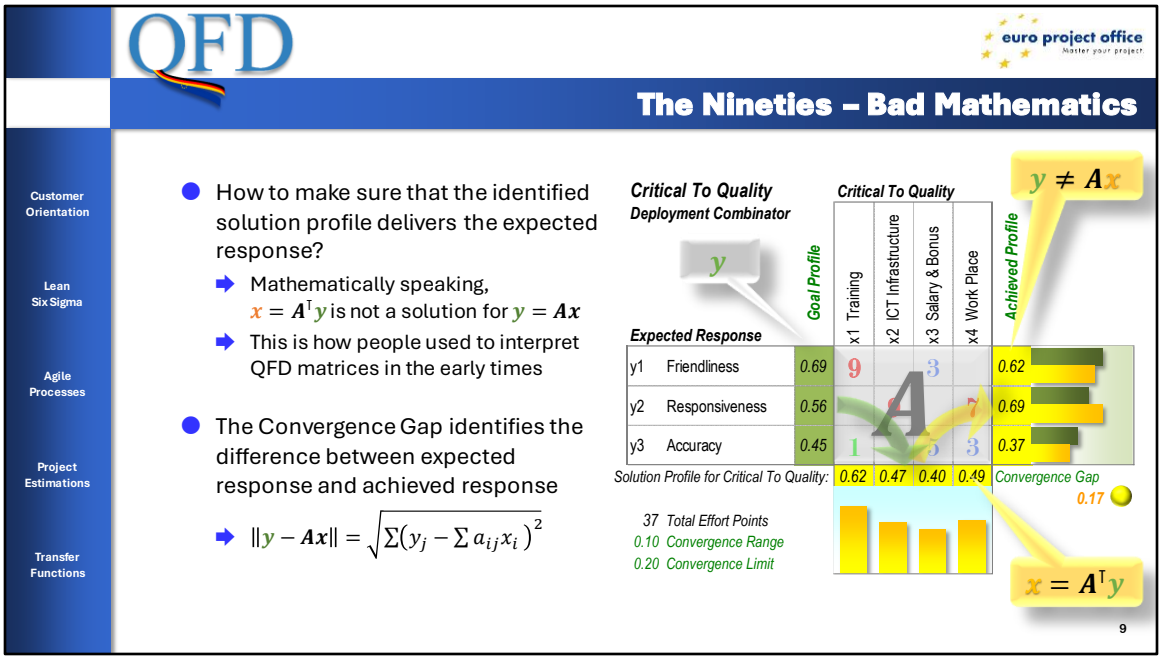
Ishikawa diagrams (also called fishbone diagrams, herringbone diagrams, cause-and-effect diagrams, or Fishikawa) are causal diagrams created by Kaoru Ishikawa (1968) that show the causes of a specific event, or response of a system. Common uses of the Ishikawa diagram are product design and quality defect prevention to identify potential factors causing an overall effect. Each cause or reason for imperfection is a source of variation.

In case the single fish becomes a huge fish swarm, matrices proved to be handier. This was the advent of *Quality Function Deployment* (QFD).

Different causes, or controls, produced effects, or responses, at different strengths. Often, the strength, or correlation, is not fixed but can be varied, as in this example, where four controls are used to produce three responses in a call-in help desk system. You can train the people, improve ICT infrastructure, increase salary and bonus depending on success, or propose better workplaces, for improving call-in service.



Yoji Akao (赤尾 洋二, Akao Yōji, 1928 – October 24, 2016) was a Japanese planning specialist recognized as the developer of Hoshin Kanri (a strategic planning methodology). With the late Shigeru Mizuno, he developed Quality Function Deployment (a group decision making technique). Akao and Mizuno also co-founded the ***Quality Function Deployment Institute***: a non-profit organization dedicated to dissemination and advancement of QFD.

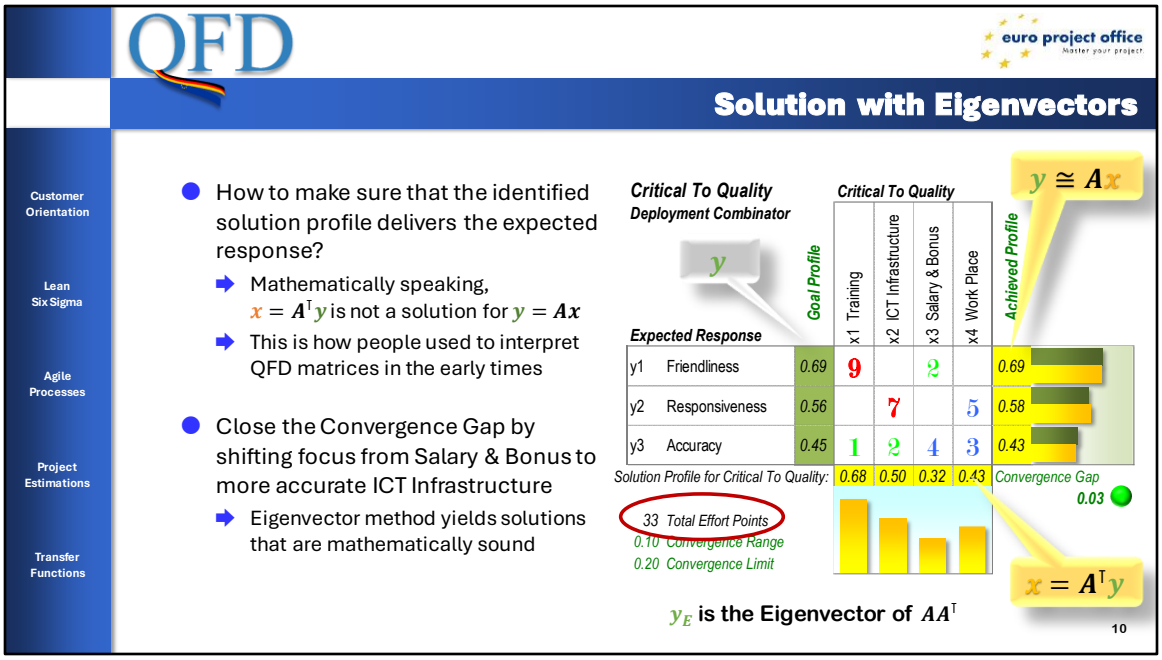


Traditional QFD matrices were usually prepared by expert teams agreeing on the coupling between controls $\boldsymbol{x}$ and responses $\boldsymbol{y}$. Old QFD used the following "bad mathematics" to solve equation $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$:

$$\boldsymbol{x} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}$$

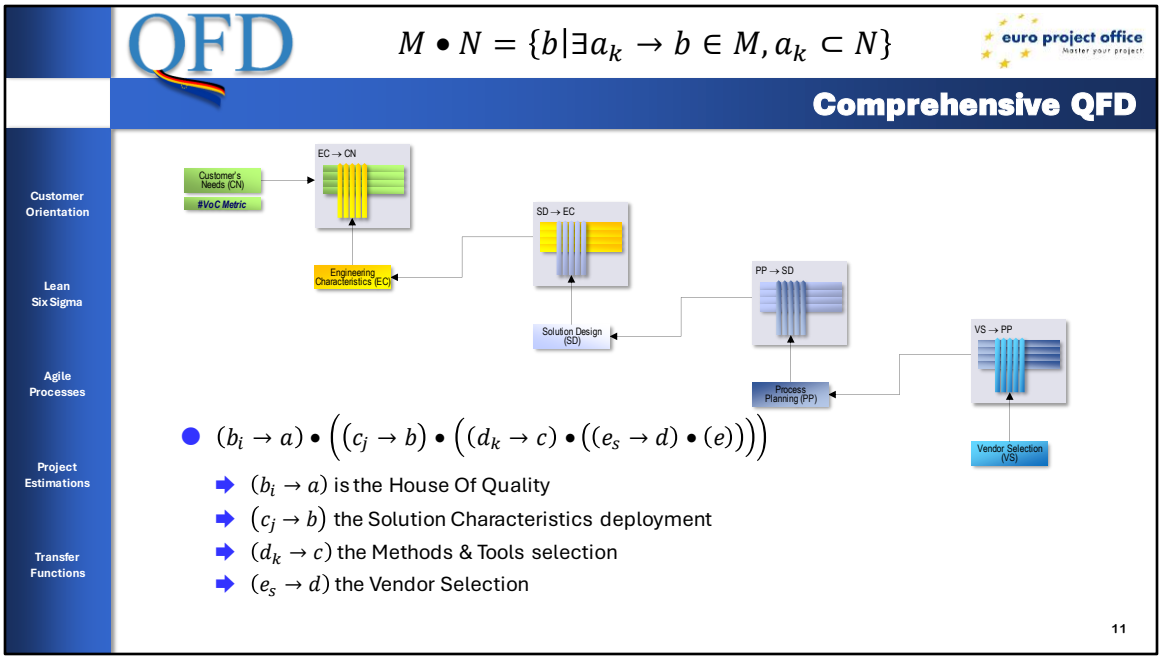
$$x_i = \sum a_{ji} y_j$$

where $\boldsymbol{A}^{\mathrm{T}} = (a_{ji})$ is the transpose of matrix $\boldsymbol{A} = (a_{ij})$. The graphic shows $\boldsymbol{x} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}$. Obviously, $\boldsymbol{x} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}$ is not a solution for $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$, but this approximation yielded good-enough solutions for the relative importance of controls, in many cases. The reason for this is that the initial step for numerically solving $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ is $\boldsymbol{x}_0 = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}$, and that experienced QFD moderators could "read a matrix" to assess whether the approximate solution $\boldsymbol{x} = \boldsymbol{x}_0$ was good enough.



Thanks to the Eigenvector method, it becomes possible to detect missing solution elements such as here the wrong focus on Salary & Bonus instead of improving the ICT Infrastructure.

Note that the total effort, i.e., cost, also decreased. As small this example might be, it shows quite clearly that QFD quite often does not behave as expected but yields surprising results, same as Saaty's AHP does, from where we inherited the Eigenvector method.



A comprehensive QFD deployments in the traditional manner is a cascading application operation between knowledge as follows:

$$(b_i \rightarrow a) \bullet \left((c_j \rightarrow b) \bullet \left((d_k \rightarrow c) \bullet ((e_s \rightarrow d) \bullet (e))\right)\right)$$

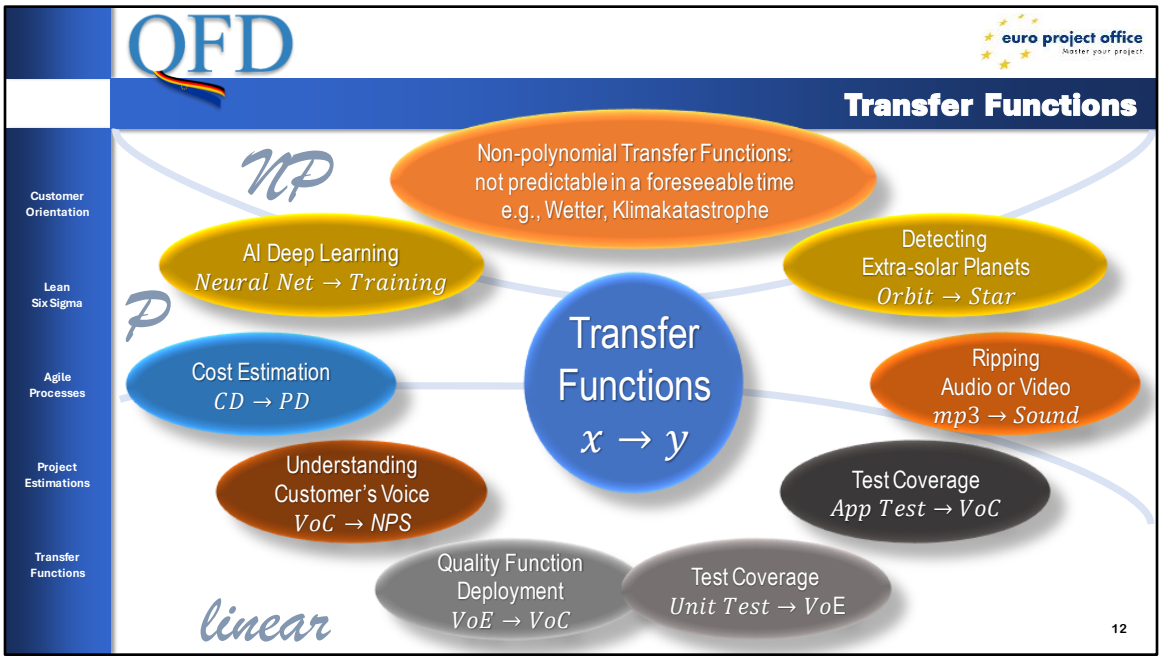
where $(b_i \rightarrow a)$ is the House Of Quality, $(c_j \rightarrow b)$ the Solution Characteristics deployment, $(d_k \rightarrow c)$ the Methods & Tools selection, $(e_s \rightarrow d)$ the Vendor Selection; and where (e) is the set of vendors, (d) the methods & tools, (c) the solution design, (b) the solution characteristics, and (a) the customer's needs.

Using the abbreviations in the above figure, the arrow scheme equation reads as

$$(EC_i \rightarrow CN) \bullet \left((SD_j \rightarrow EC) \bullet \left((PP_k \rightarrow SD) \bullet ((VS_s \rightarrow PP) \bullet (VS))\right)\right)$$

If you think this simplistic example is complicated, read again Akao's Big Black Book from 1990!

What matters is that the QFD process is applicable to other QFD processes. It is a combinatory algebra, with recursiveness.

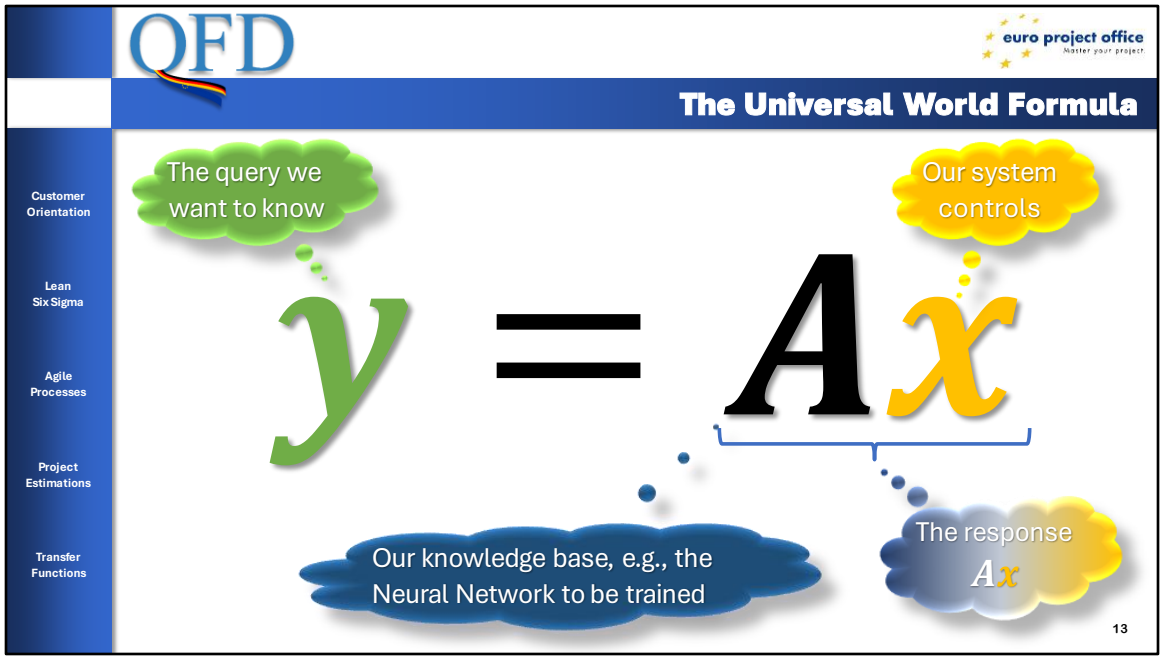


What was the most important invention of the 20th century??

The proof that the *Fast Fourier Transform* (FFT) is not NP-complete had the greatest impact on mankind, probably since the invention of cooking on fire (Cooley & Tukey, 1964). In 1977, this led to the possibility of converting audio signals into a digital code in a predictable time. Previously, sounds were analogue vibrations of the atmosphere or electromagnetic potentials caused by living beings or loudspeakers. With the FFT, it became possible to describe with digital numbers what the original vibrations were. The consequences were that chips became available that converted audio signals into storable digital code, and later also video signals. The music and entertainment industry was turned upside down by this invention. Today, computers, laptops, telephones and televisions are equipped with this technology and connected to each other via the global Internet.

The FFT did not come about by chance but was the result of years of solid research in mathematics, especially linear algebra. The FFT algorithm selects the Fourier basis functions in the functional vector space that models the corresponding signal and represents the signal by the coordinates of the unit vector in that space. This works in a functional vector space. In general, this is the principle of finding controls for a transfer function to explain an observed effect.

Welcome to the latest new addition to Transfer Functions, Deep Learning 2012!

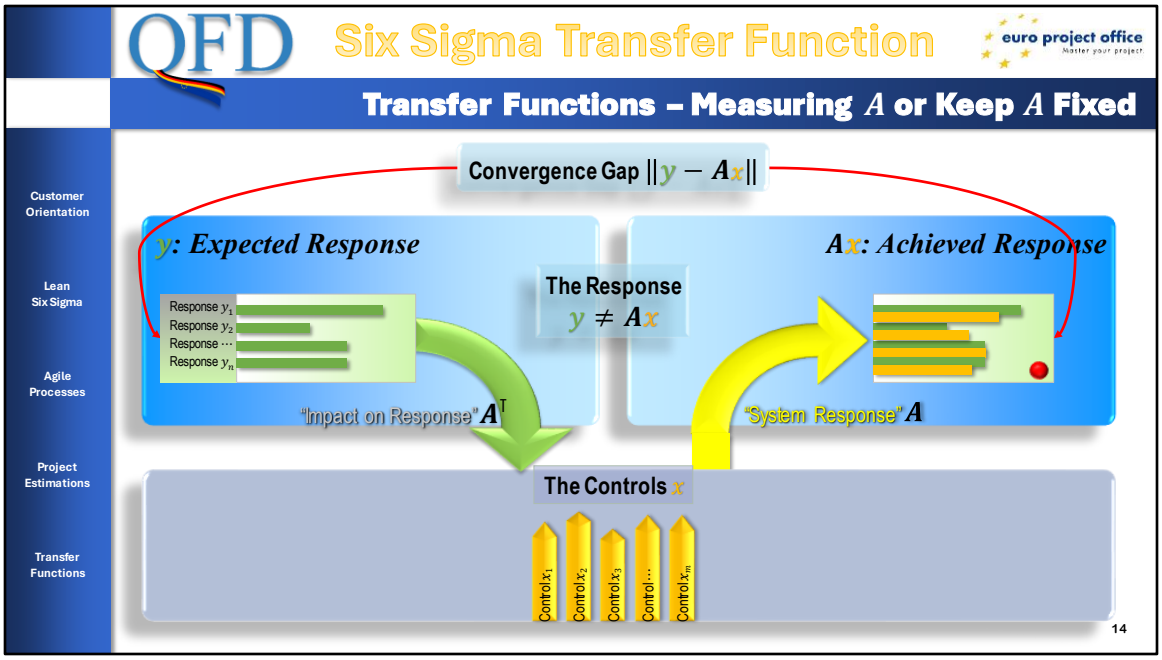


That is the magic formula. Solving the formula for the observable $\boldsymbol{y}$ and the unknown $\boldsymbol{x}$ is what fascinates us.

$\boldsymbol{A}$ is the transfer function that represents the knowledge that we must acquire first. It is not a given functionality, but what we need to learn. Sometimes, in Six Sigma, we can measure (some of the) the elements of the matrix $\boldsymbol{A}$, sometimes, in QFD, we must guess values based on expert discussions. The controls $\boldsymbol{x}$ are then an optimum solution providing the response $\boldsymbol{y}$, or something quite close.

Artificial Intelligence is also a solution to this formula. However, the goal is to train the knowledge base $\boldsymbol{A}$ with known facts about $\boldsymbol{y}$ and known responses $\boldsymbol{x}$. Then, after sufficient training, we can ask the system $\boldsymbol{A}$ by providing an input vector $\boldsymbol{x}$ to get a response $\boldsymbol{y}$. This requires a much larger matrix $\boldsymbol{A}$ because of the many possibilities that different input vectors $\boldsymbol{x}$ can generate.

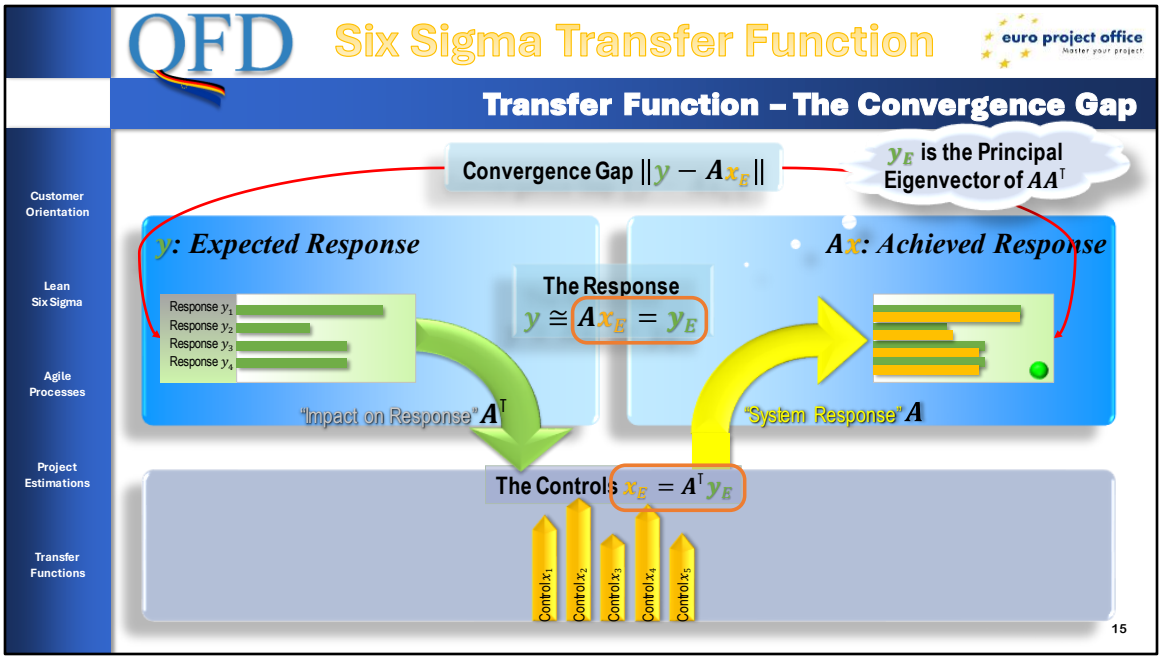
In AI, we must deal with neural networks that can be represented as sparse matrices of huge size (ChatGPT has some $40'000 \times 70'000$ neuronal cells). In good old QFD we probably can manage 12×20.



In AI, the matrices are no longer 3×4, but $30'000 \times 40'000$, for a Large Language Model for instance. The principle is still the same:

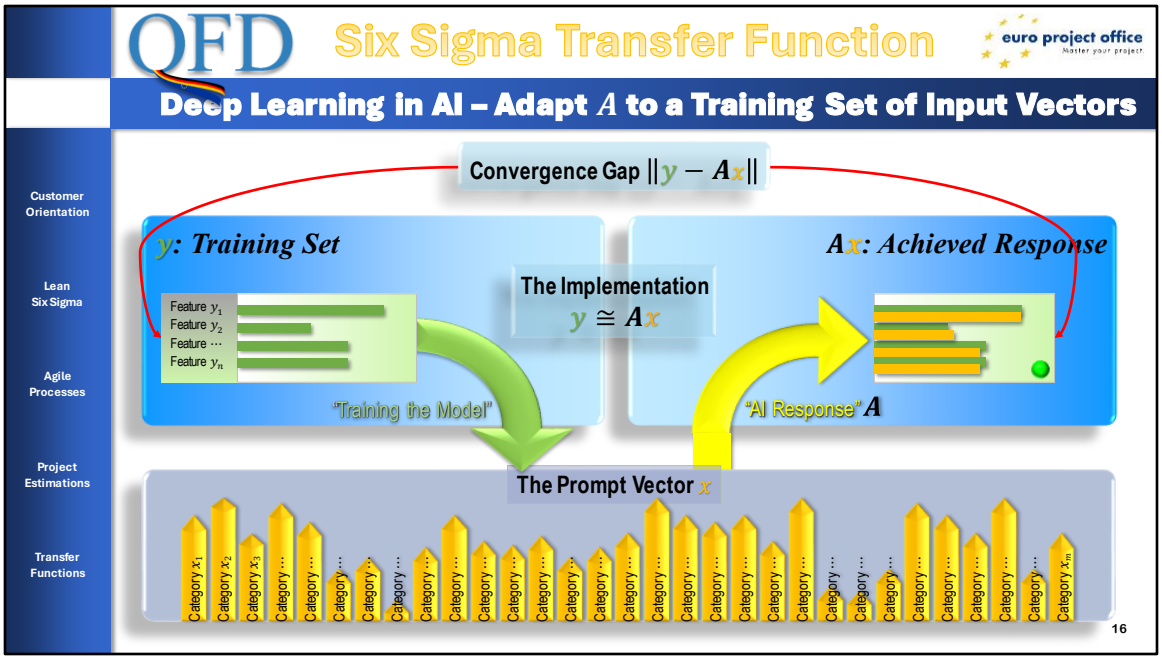
- Count frequencies of characteristics of samples provided;
- Link to objects recognizable by a Neural Net, i.e., some very large matrix;
- Apply Neural Net to queries.

If the result is unsatisfactory, improve the Neural Net by adjusting the weights of its neurons.



Now, applying transfer functions for new product development is straightforward. The intended response define the goals of a product: the "What". The controls are whatever makes the product work as intended by defining the "How".

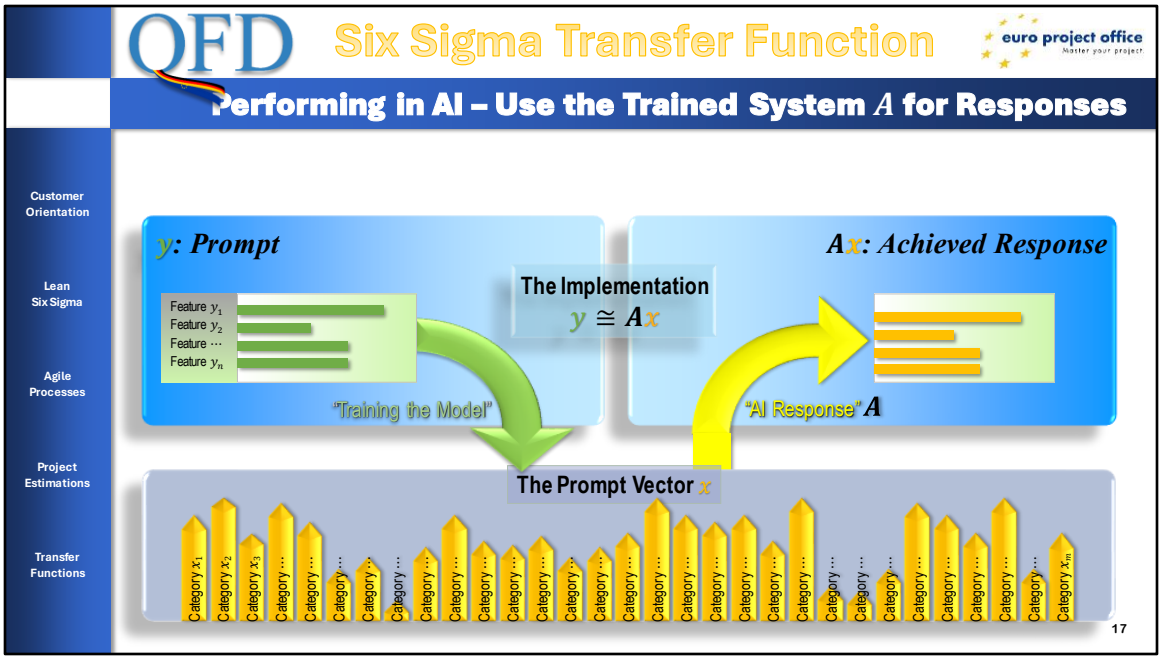
The difference between the expected response y, and the achieved response Ax, is called the *Convergence Gap*. It is the Euclidian length of the vector difference between the achieved system response Ax and the expected goal response y.



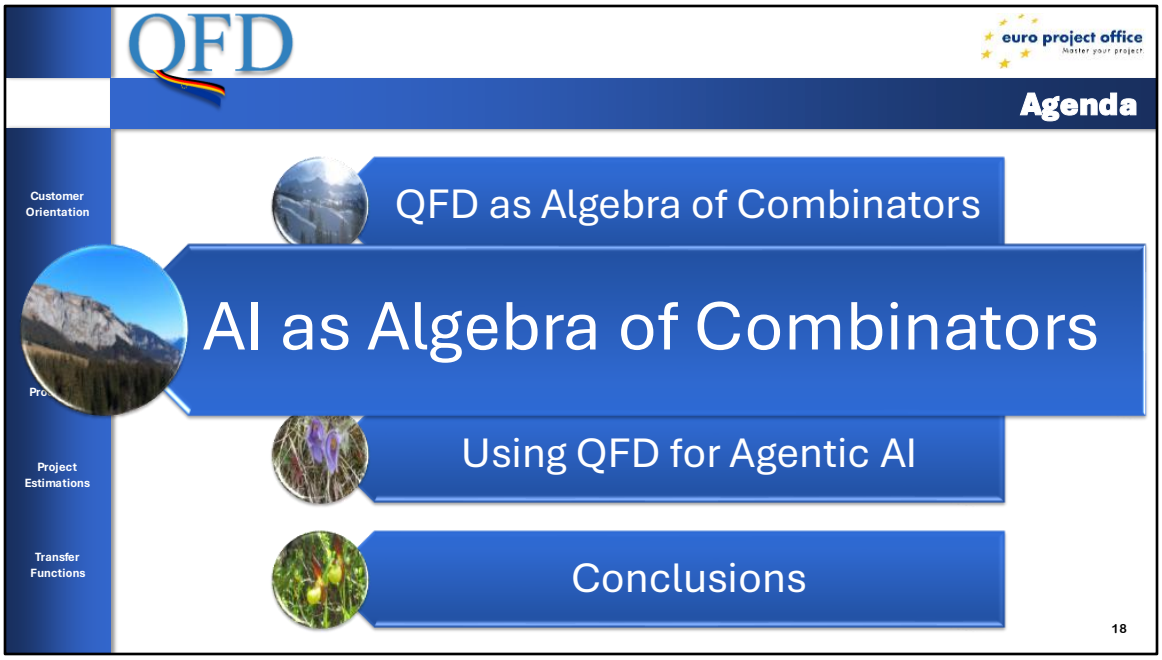
The principle of ***Deep Learning*** is similar between humans and machines – it means means that setting a transfer function $\boldsymbol{A}$ which, given an input vector $\boldsymbol{x}$, produces a response $\boldsymbol{Ax}$ that is sufficiently close to the correct answer; correct in the sense that it matches the features of the training set.

For an ***Artificial Neural Networks*** (ANN), which here corresponds to the neural network $\boldsymbol{A}$, the weights of the nodes is adjusted so that the response vector $\boldsymbol{Ax}$ matches the training features $\boldsymbol{y}$ as best as possible. The difference between $\boldsymbol{y}$ and $\boldsymbol{Ax}$ is called ***Convergence Gap***.

This difference is calculated by the Gauss' method of the least squares.



Performing means apply the trained controls, i.e., the weights of the nodes of the neural network, and produce an answer. A proof that the response is correct is not available immediately; it must be compared to the environment, or to some other source of evidence.



- QFD as Algebra of Combinators
- AI as Algebra of Combinators
- Using QFD for Agentic AI
- Conclusions

QFD

euro project office
Master your project

Counting Frequencies

Customer Orientation

Lean Six Sigma

Agile Processes

Project Estimations

Transfer Functions

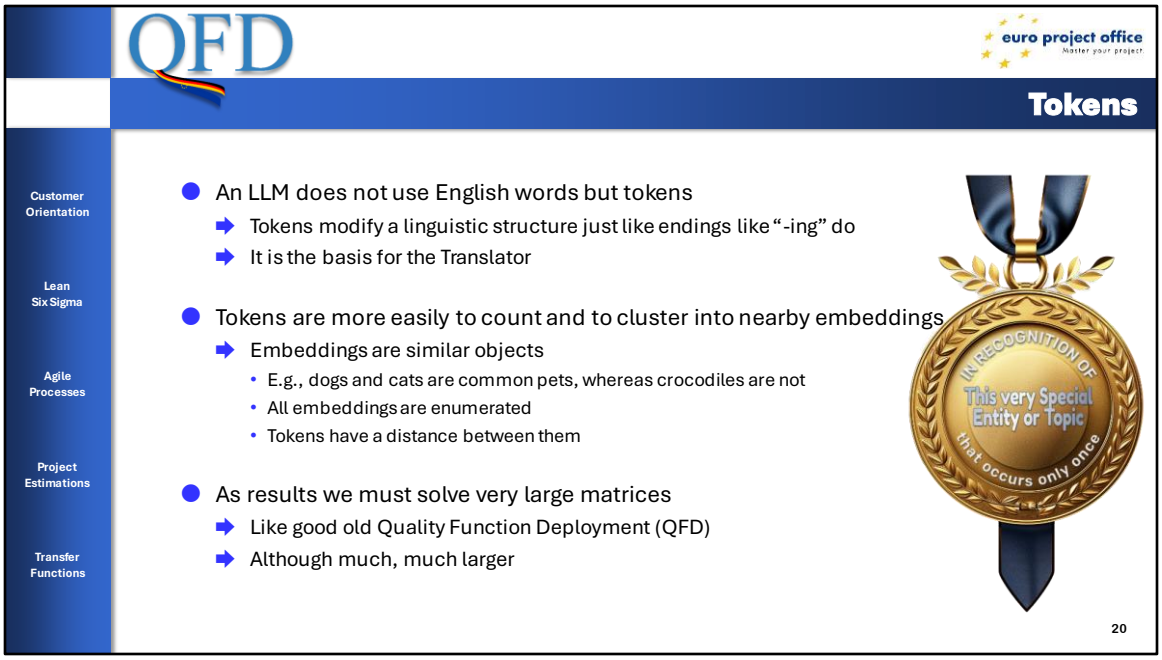
- **Large Language Models** (LLM)
  - Generative Pretrained Translator
  - Pretrained by counting words and their sequence
    - It understand nothing
  - 80° Temperature
    - Means that it selects randomly among the 20% best matches
    - Otherwise, LLM become boring
- LLMs can be used to provide a flawless, elegant English
  - And a few other languages as well, including Kölsch and even Schwyzertüüsch
  - "Daach zosamme!" or «Guete Morge zäme»

19

The basics of AI is the same as with QFD: we count frequencies to get the weight of some component.

LLM counted words in many different languages, mostly from the web. With that count, it can calculate what words belong together, their context, and how often word 1 is followed by word 2, and so one. This yields stylish but meaningless phrases.

Such a probability counts is called a ***Large Language Model*** (LLM).

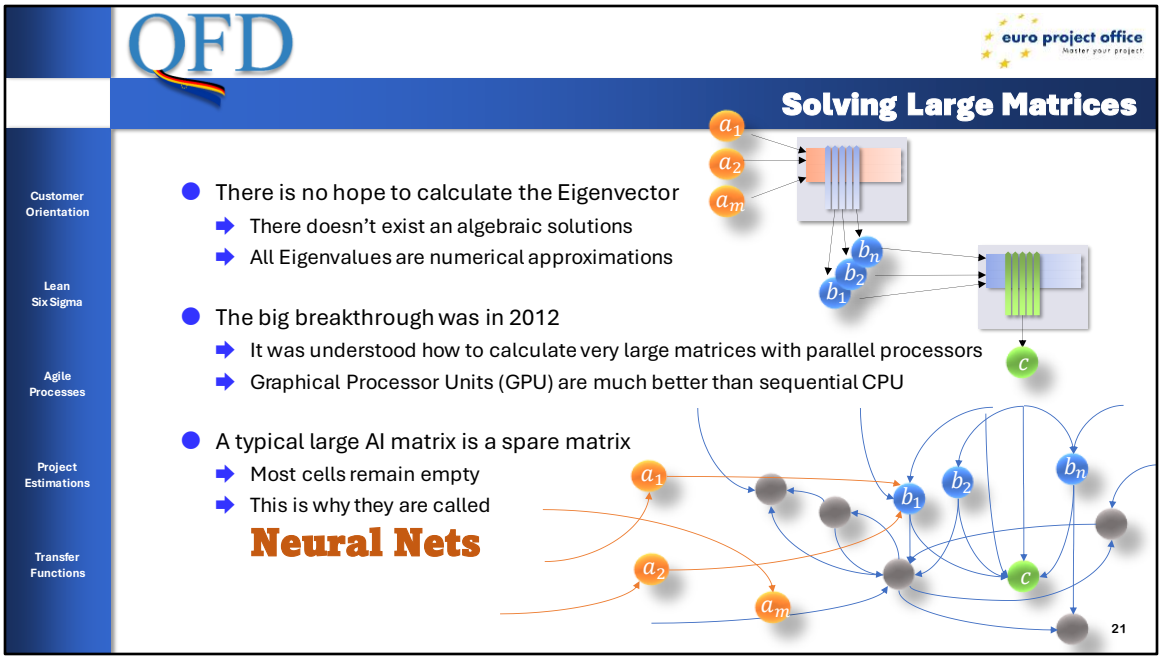


an LLM uses tokens, not English words. For translations, this makes things considerably easier.

On the other hand, to represent a simple but general English sentence, you need up to 40'000 tokens. These tokens must be lined up with their context, embedded with similar objects, and processed accordingly.

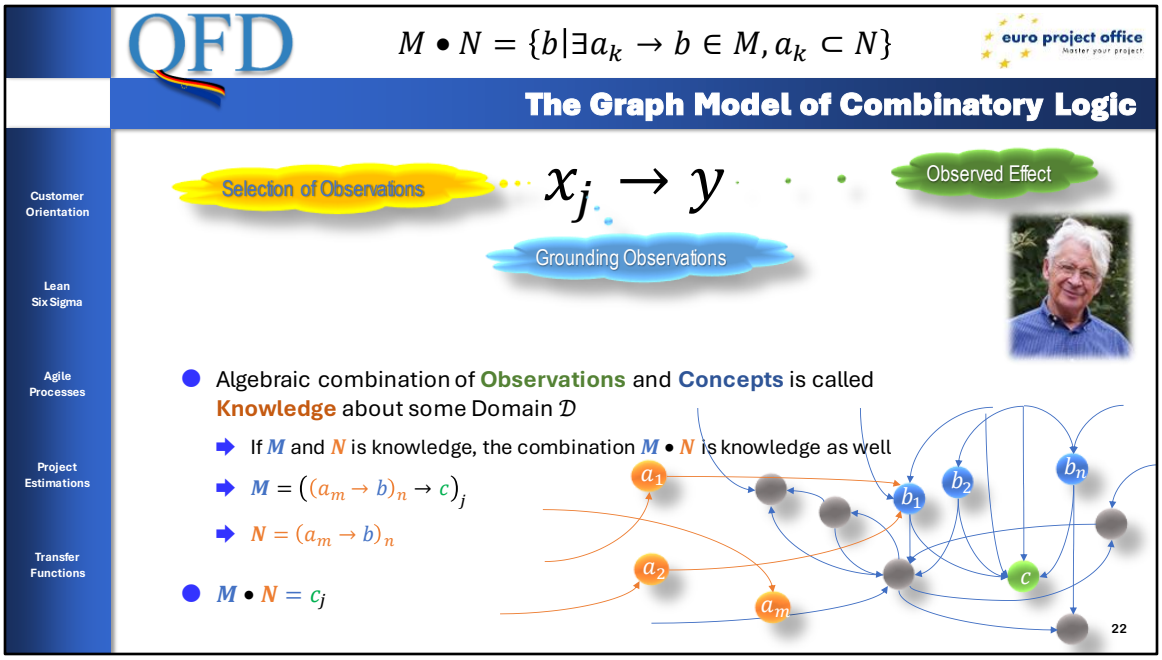
Since all tokens are enumerated, all context become vectors, and we end up in vector space with 40'000 dimensions. To work with vectors, you need linear matrices, and they get quite big, with millions of cells.

These are sparse matrices, because not all objects in the real world, and not all words in natural language, interact together.



Until 2012, it was quite unclear how to calculate such large mathematical objects. While Neural Nets were known since at least 80 years, to work with them and let them produce reasonable results, was impossible.

Engeler and Scott found that Neural Nets yielded a ***Graph Model of Combinatory Logic***, and this was interesting because combinatory logic provided a foundation for human knowledge that didn't use quantifiers. Thus, it was possible to do sensible things without the need to phrase sentences such as "For all X, it holds some statement F(X)". We know since Russel (Antinomy, 1902) that such phrases always ran us into difficulties, contradictions in logic and mathematics, and deadly prejudices in society.



New knowledge, represented by arrow terms, can be generated by combining existing knowledge with new observations. Let $\boldsymbol{M}, \boldsymbol{N} \in \mathcal{G}(\mathcal{L})$. Then application of $\boldsymbol{M}$ to $\boldsymbol{N}$ is defined by

$$\boldsymbol{M} \bullet \boldsymbol{N} = \{b | \exists a_i \rightarrow b \in \boldsymbol{M}, a_i \subset \boldsymbol{N}\}$$

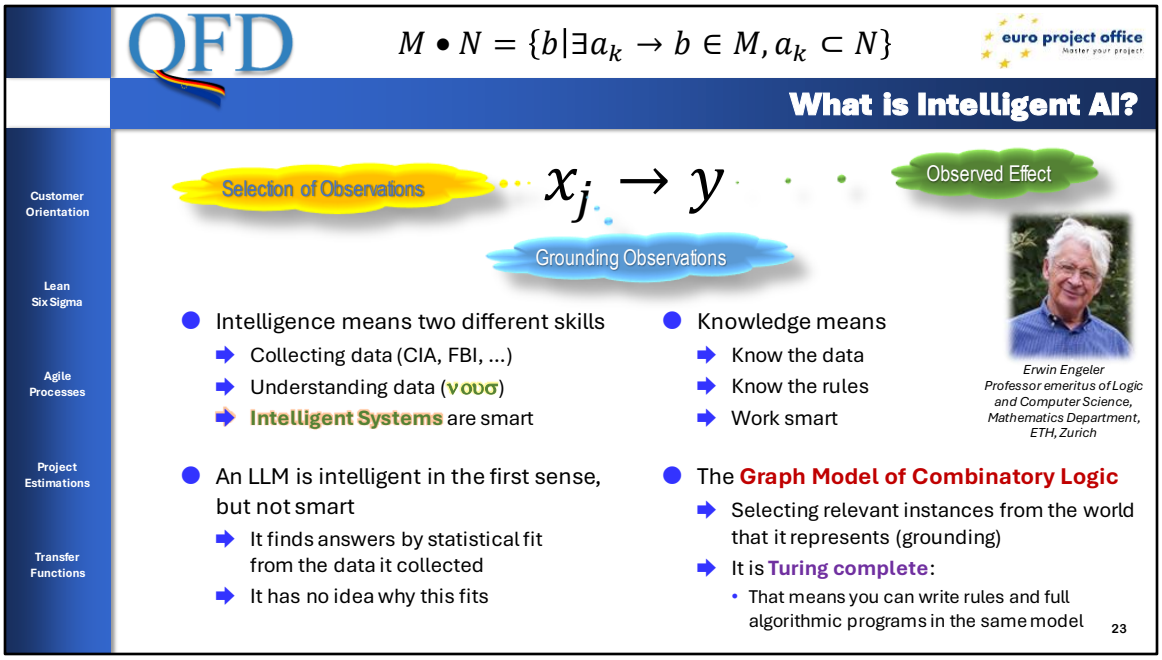
In case of $\boldsymbol{M}$ as new observations, and $\boldsymbol{N}$ as existing knowledge, this represents the selection operation that chooses those observations $(b_i \rightarrow a)_j$ from knowledge set $\boldsymbol{M}$ that are extend current knowledge $\boldsymbol{N}$. The definition applies to all higher–level $\mathcal{G}(\mathcal{L})$ terms.

Maybe a little bit complicated, but effective. The approach originates from intuitionistic and constructivist logic, avoiding the famous undecidability trap.

The Graph Model of Combinatory Logic was identified by Dana Scott, an American logician who is the emeritus Professor of Computer Science, Philosophy, and Mathematical Logic at Carnegie Mellon University, and Erwin Engeler, retired Professor ETHZ, in the 1970s.

In 2019, Engeler described in his seminal paper "Neural algebra on 'how does the brain think?'", Theoretical Computer Science, 777, 296-307. how to model biological neuronal behavior with the Graph Theory.

"Grounding observations" means that AI classes of objects are linked to objects in the real world (Zhong, V. et al., 2022. Improving Policy Learning via Language Dynamics Distillation, Cornell University: arXiv:2210.00066v1 [cs.LG]).

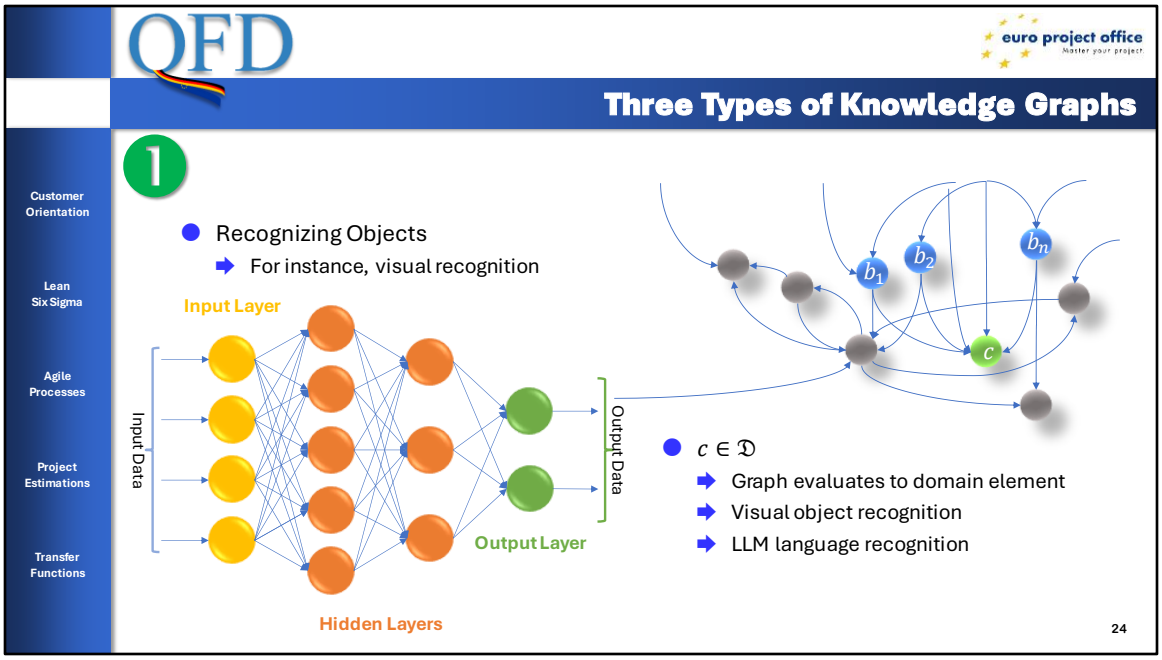


The problem with AI is, AI is not intelligent. At least not intelligent in sense of smart. Smart Things in IoT are often called Intelligent Systems, but today less popular than all the AI engines.

Νους is what Homer attributed to the android robots (read as female by Homer) of Hephaistos: they were not only able to understand what their master god was saying, but also what he meant. These androids were able to make a new suit of armor for Achilles after his friend Patroclus lost it to Hector.

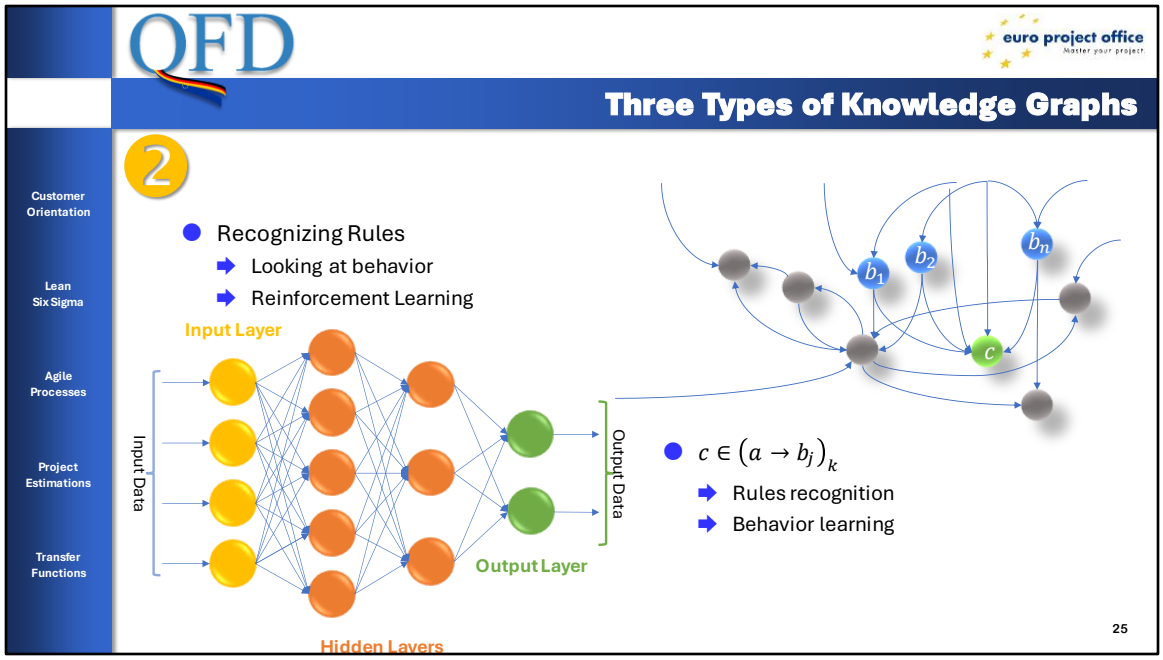
Engeler and Scott found around 1980 that there is a graph model of combinatory logic that is Turing-complete and thus can be said representing knowledge in an algebraic way. These arrow terms represent edges in a neural network, between many observations x_j and some observed effect y.

The graphs represent neural networks and therefore the model has some impact on modern AI.



There are three different types of knowledge graphs.

The first type 1 is about graphs that remember objects, visual or textual. The neural network created by this graph can be used to recognize objects. This means a graph evaluates from some given input and finds out what this object is all about: name, properties, use, description, and whatever can statistically be captured about such an object.



Another application for neural networks is remembering rules. Rules are also arrow terms but with a temporal meaning of the arrow. A type 2 knowledge graph tries to remember rules, for instance for learning behavior or preferences of a skilled user.

QFD

euro project office
Master your project

Three Types of Knowledge Graphs

3

Customer Orientation
Lean Six Sigma
Agile Processes
Project Estimations
Transfer Functions

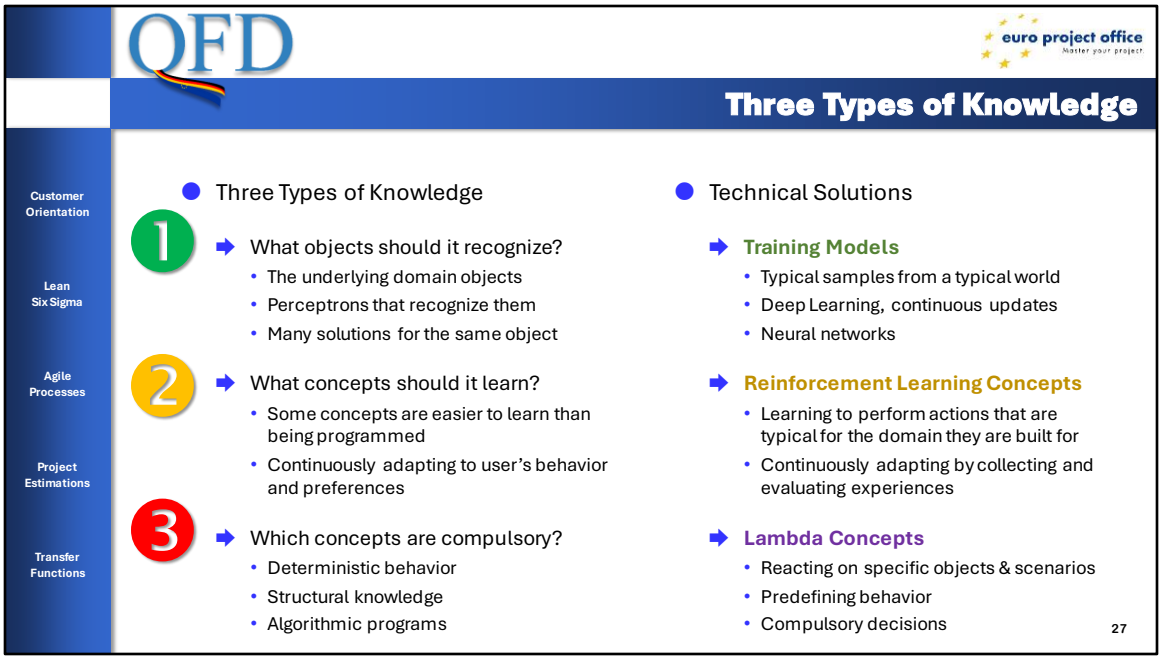
- Closed Loop Graphs
  - Called Lambda Concepts
  - Algorithmic behavior
  - Traditional Programs
  - **Logical Derivation**

- Substitution of arrow terms
  - $\mathbf{I} = a_1 \to a$
  - $\mathbf{K} = a_1 \to \emptyset \to a$
  - $\mathbf{S} = \left(a_i \to \left(b_j \to c\right)\right)_1 \to \left(d_k \to b\right)_i \to \left(a_i + b_{j,i} \to c\right)$
  - $\lambda f.\mathbf{K}f = \mathbf{S}\lambda f.\mathbf{K}\lambda f.f = \mathbf{S}(\mathbf{KK})\mathbf{I}$

26

Type ❸ knowledge graphs are quite different. As graphs, they are closed loop. Moreover, since knowledge can be empty, some of these graphs are rather difficult to visualize. Or how would you display the combinator $\boldsymbol{K}$?

Type ❸ knowledge graphs are well-known as algorithmic programs. Although their name "Lambda-Concept" points at something very theoretical, they represent the Python-programs that actually run all these AI-Engines such as Agents, Bots, Cathegorizer, etc. that appeared in the last years, since the inception of Large Language Models (LLM).



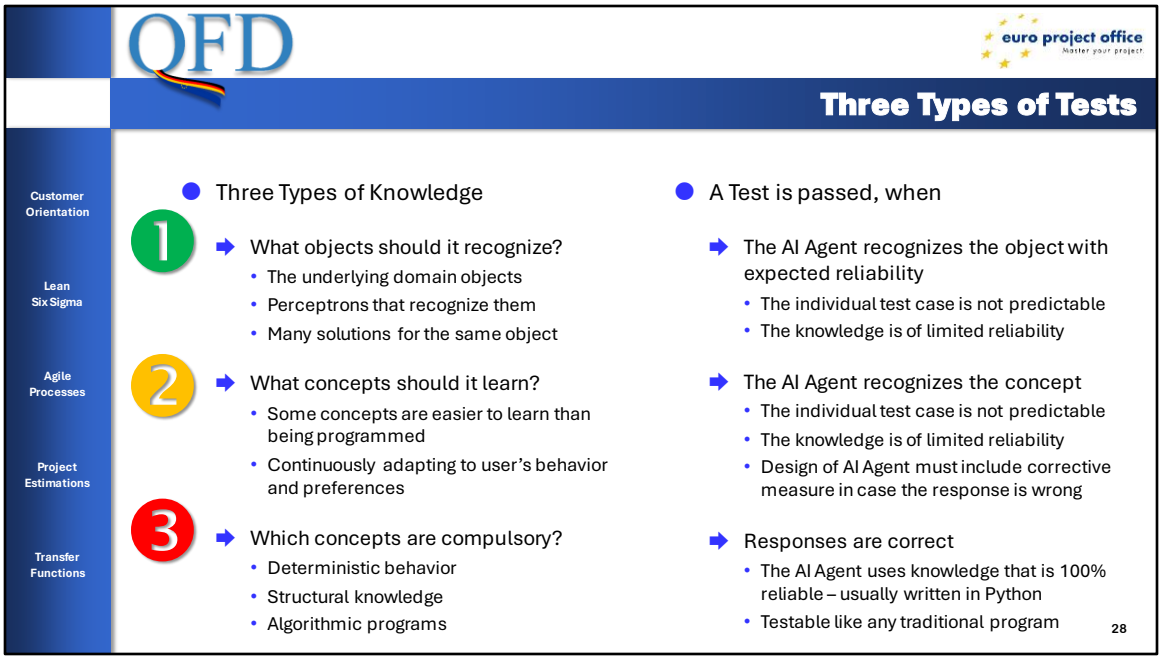
We therefore need to specify not only the objects that our intelligent systems must be able to recognize, but also the compulsory concepts, and how it should learn new concepts.

We classify thus requirements in three categories that correspond to three different technical solutions. Compulsory concepts can be identified with User Stories for traditional programs (type 3) with deterministic functionality, while type 1 and type 2 requirements have a lower reliability because they might reflect an LLM hallucination.

Two kinds of knowledge are observational and evaluate to recognized objects or concepts from the base domain; they can be learned by perceptrons; one kind of knowledge is processual and makes it possible to program behavior and object recognition.

There are always a multitude of solutions. Knowledge is not reducible to some normal form; neither Combinatory Logic nor Requirements.

This is a consequence of Gödels undecidability theorem.



Tests are passed depending on the type of knowledge. In case of the two that are observational and evaluate to recognized objects from the base domain or observed rules about the domain, Tests have a statistical variability. Test Cases can be true, false true, false wrong, or wrong.

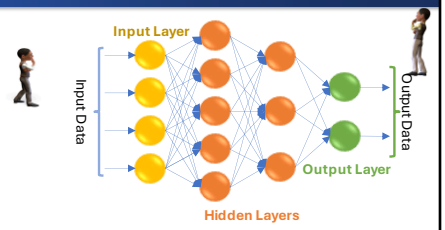
Only the tests for the closed loop type of knowledge have a well-defined result. This is how we test AI Agents that are relevant to business, or security, or safety.

QFD

euro project office
Master your project

Explainable AI (XAI)

Customer Orientation
Lean Six Sigma
Agile Processes
Project Estimations
Transfer Functions



- A big problem for AI is that nobody knows what happens in hidden layers
- It is unlikely that hidden layers implement the most straightforward arrow schemes
  - In contrary, combinatory logic has no normal form
  - There exist many ways to find a valuable result, even for the same result
- Regulators try to impose rules for training sets
  - It is very difficult to assess quality of a training set, because knowledge is always limited
- Current explainability techniques
  - SHAP (SHapley Additive exPlanations) is a game-theoretic approach
  - LIME (Local interpretable model-agnostic explanations) is a method that fits a surrogate glass-box model around the decision space of any black-box model's prediction
  - A lot more, all based on statistical correlations, not on reasoning

29

The problem with explainable AI is certainly that the hidden layers in a perceptron remain hidden. They are not labeled as with comprehensive QFD.

Nevertheless, there exit many explainability techniques. However, all have the problem that they use statistical correlations as a basis for reasoning. That cannot go well because of Gödel's undecidability theorem. But it's probably better than nothing.

These tools shall create trust and block further investigations.

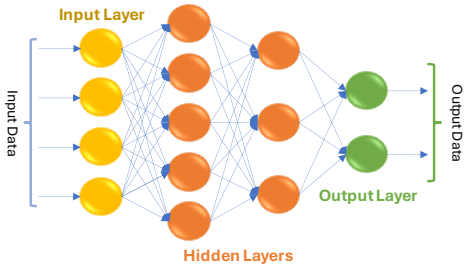
QFD

euro project office
Master your project

Intelligent Systems are Based on Controlling Combinators

Customer Orientation
Lean Six Sigma
Agile Processes
Project Estimations
Transfer Functions

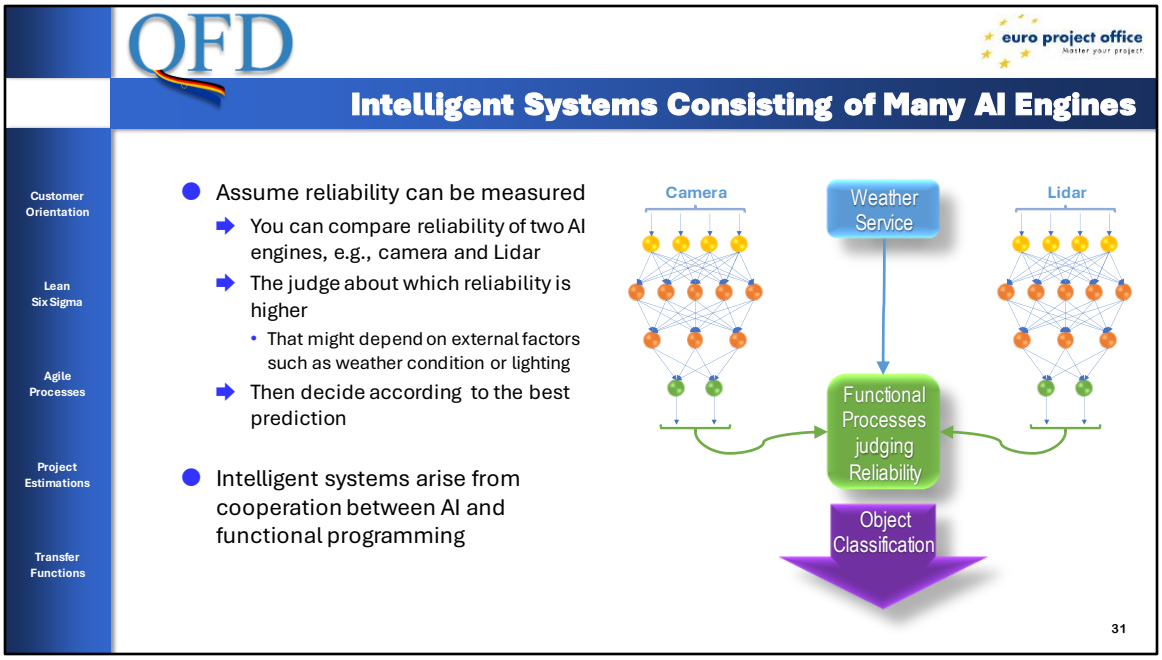
- **Controlling Combinators** C do loop and augment knowledge X under control
  - Controlling Combinators are of type ❸
  - Focusing $X_{i+1} = C \bullet X_i, i \in \mathbb{N}$
  - Control Sequence $X_0 \subseteq X_1 \subseteq X_2 \subseteq \cdots$
  - The amount of available knowledge is measured as **Reliability**
  - Stop looping when approximation to correct response is good enough
- Neural Networks in the brain do loop!
  - Controlling Combinators keep looping programs under control
- Deep Learning by multi-layered perceptrons (MLP) is ordered
  - No loops from input layer to output layer



30

However, there are open questions. Humans must likely use neural networks with feedback loops. AI does not.

What could happen if AI learns how to use feedback loops? **Controlling Combinators** (Engeler 2019, "How does the brain think") are crucial for further developing artificial intelligence and better approximate human intellectual capabilities.



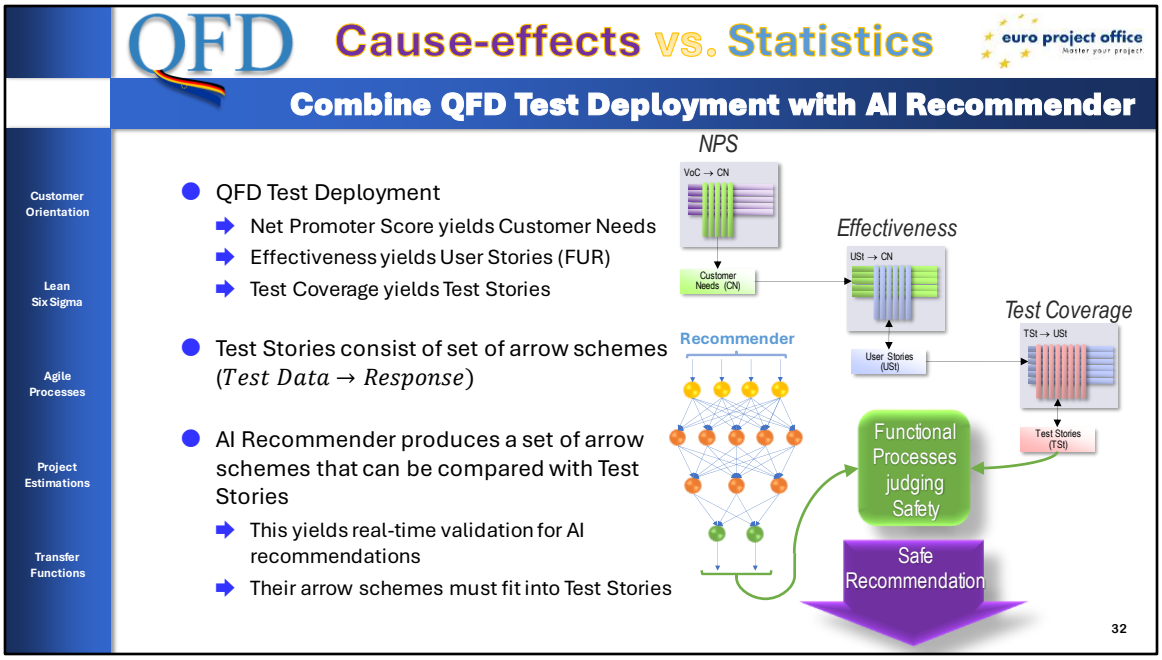
It is also possible to add more than one AI engine to an intelligent system, compare results and go forward with the most reliable one. Insufficient training, biases, and hallucinations therefore would become detectable.

This slide shows an example of an intelligent system design that relies on two separate visual recognition engines analyzing the same scenario, one through a camera and the other through a Lidar.

Such an architecture requires that the reliability of each artificial intelligence engine be known, under certain conditions, such as weather. In this way, the intelligent system can explain why it selected one or the other response.

Obviously, if both AI-engines produce an identical response, this increases overall reliability of the response of the intelligent system quite a bit.

The graph model delivers the metrics for defining controlling combinators by inclusion, and it also allows to combine knowledge and thus reliability correctly.

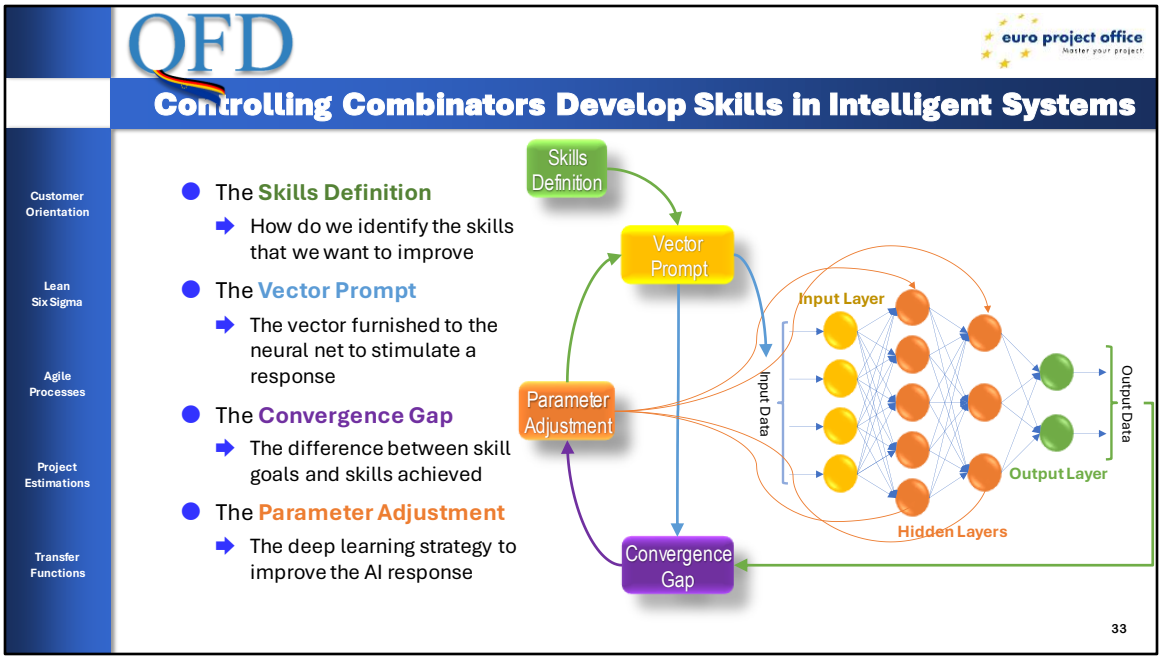


This is yet future work, just an idea, not yet thought through.

Arrow schemes also work well as representation for test stories.

The QFD Test Deployment is an upfront investment that defines the range of acceptable responses. Thus, all you must do is use the same tokens for QFD and for AI and compare them.

More sophisticated combinations are also possible, for instance, combining the Recommender with a Controlling Combinator for continuous learning.



These functional processes depend on two parameters: the skills definition and the feedback from some external authority. This includes the convergence gap definition that must be provided externally from some empirical source. The convergence gap is the gap between skills and external expectations.

Then the intelligent system can do the fine-tuning autonomously by gaining control over certain capabilities. It needs feedback from attractors in the external world, and these attractors are implemented by functional processes that compute the convergence gap against some external reference and feed back the result to the AI engine for improvement.

Thus, intelligent systems are a joint venture between AI engines and traditional functional processes.

Obviously, the **Convergence Gap** creates the problem how to calculate it.

.

QFD

euro project office
Master your project

The Cause-Effect Concept Generator (CECG)

Customer Orientation

Lean Six Sigma

Agile Processes

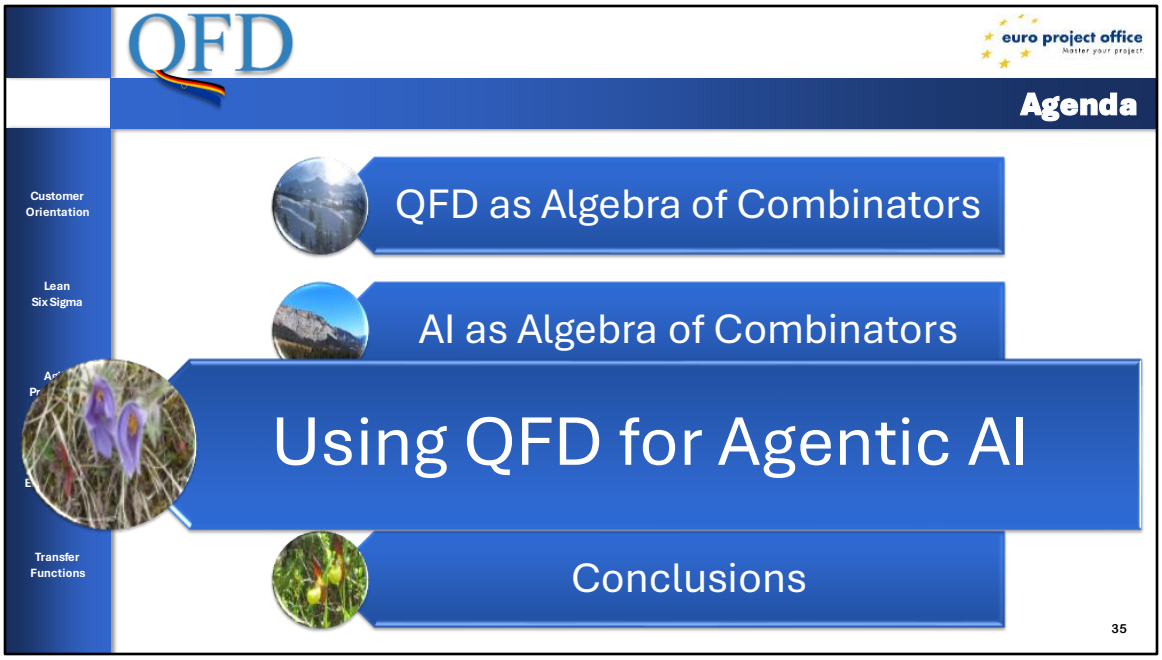
Project Estimations

Transfer Functions

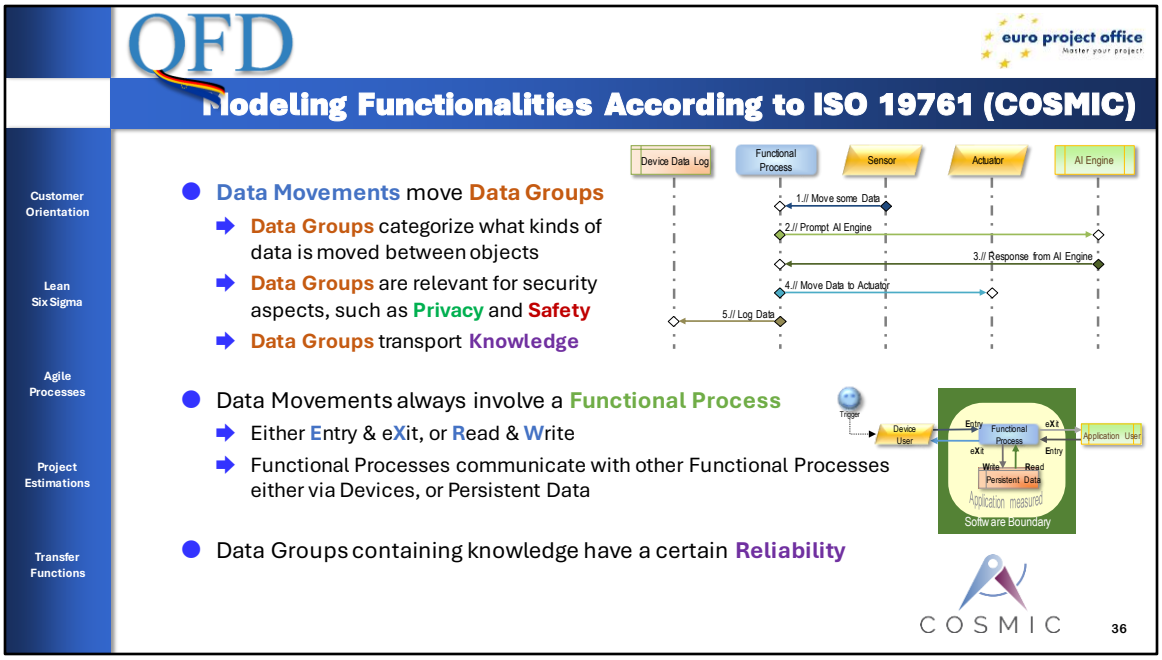
- An **Intelligent System** is a hybrid system between Symbolic AI and LLMs that checks responses with a **Cause-Effect Concept Generator** (CECG)
  - A CECG generates rules derived from some rule set, using the same tokens as the LLM
  - Because tokens have a distance, building a convergence gap calculator is relatively easy
  - The CECG uses mathematical logic to derive valid combination of rules
- 15 years ago, the world was full of IoT Smart Systems, also called Intelligent Systems
  - However, they used no AI Engines
  - All was algorithmic
  - Type ❸ requirements
- An LLM is not smart
  - A CECG can be smart
  - It's classical, symbolic AI
  - With smart search

15 years ago, the world was full of IoT Smart Systems, also called Intelligent Systems. This seems forgotten, nowadays where there is AI. However, smart systems followed predominantly type ❸ requirements, i.e., they were algorithmic. Surprisingly, you cannot solve all problems with deterministic algorithmic solutions (because of Gödel's Incompleteness Theorem).

Nowadays, we use CECGs to keep LLMs under control.



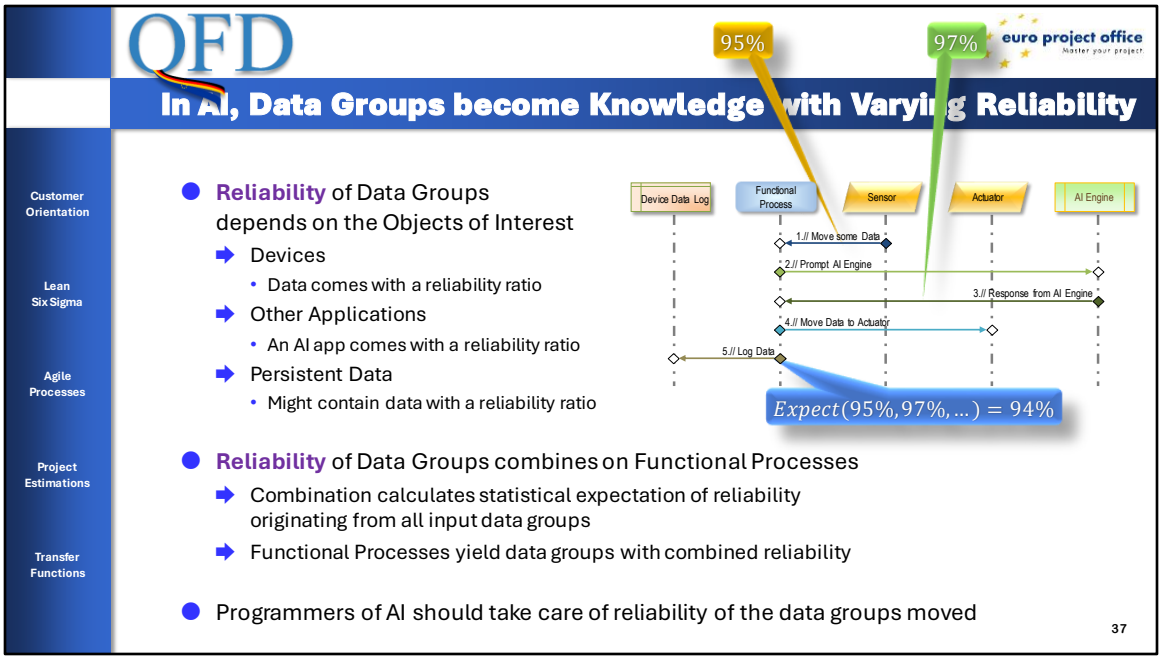
- QFD as Algebra of Combinators
- AI as Algebra of Combinators
- Using QFD for Agentic AI
- Conclusions



The ISO/IEC 19761 international standard exists since more than 20 years and defines a way of modeling the functionality of software. Neither quality aspects are considered, nor implementation details. The model is equally valid for own code, for services originating from elsewhere (e.g., the cloud) including microservices, and for the expected behavior of intelligent devices or services. It measures functionality by counting ***Data Movements*** (Entry & eXit, or Read & Write), that move ***Data Groups*** between ***Objects of Interest***. Data groups are always moved from or to a ***Functional Process;*** objects performing tasks according to ***Functional User Requirements*** (User Story).

Functionalities are expressed as User Stories in the Agile context. According to ISO 14143, functional requirements must meet a defined level of granularity. ***User Stories*** name the device user, the requestor, and describe what a functional process shall do, specifying data stores, devices, and other applications needed.

Creating a ***Data Movement Map*** from a set of user stories is an easy way of modeling functionalities with defined granularity.



Reliability ratios can be assigned to *Objects of Interest* – Devices, and Other Application (AI), reflecting the degree of trustworthiness, of data originating from them. Persistent Data objects, that might contain untrusted data resulting from some knowledge-dependent action, or from a sensor, has also a limited reliability ratio, but the persistent data object does not change reliability. These objects of interest dynamically produce data groups with varying, limited reliability. Although reliability is measured commonly by a percentage number, it is a standard deviation, not a linear reliability average.

The Functional Processes combine these reliabilities, by combining insecurities inherent with knowledge-based actions. Thus, in our sample, the minimum reliability of data groups originating from the (blue) functional process is the combination of the reliability of all incoming data.

If data is persistently stored, it retains its reliability.

QFD

euro project office
Master your project

A Simple AI Agent for Implementing Business Processes

Customer Orientation
Lean Six Sigma
Agile Processes
Project Estimations
Transfer Functions

- Our Organization needs an AI Agent that quotes customers individually based on
  - Business Rules
    - For Terms & Condition
    - Discounts
  - Price Book
- Customer Needs
  - By AHP
  - Integrating stakeholders
    - End Customers
    - Management
    - Compliance

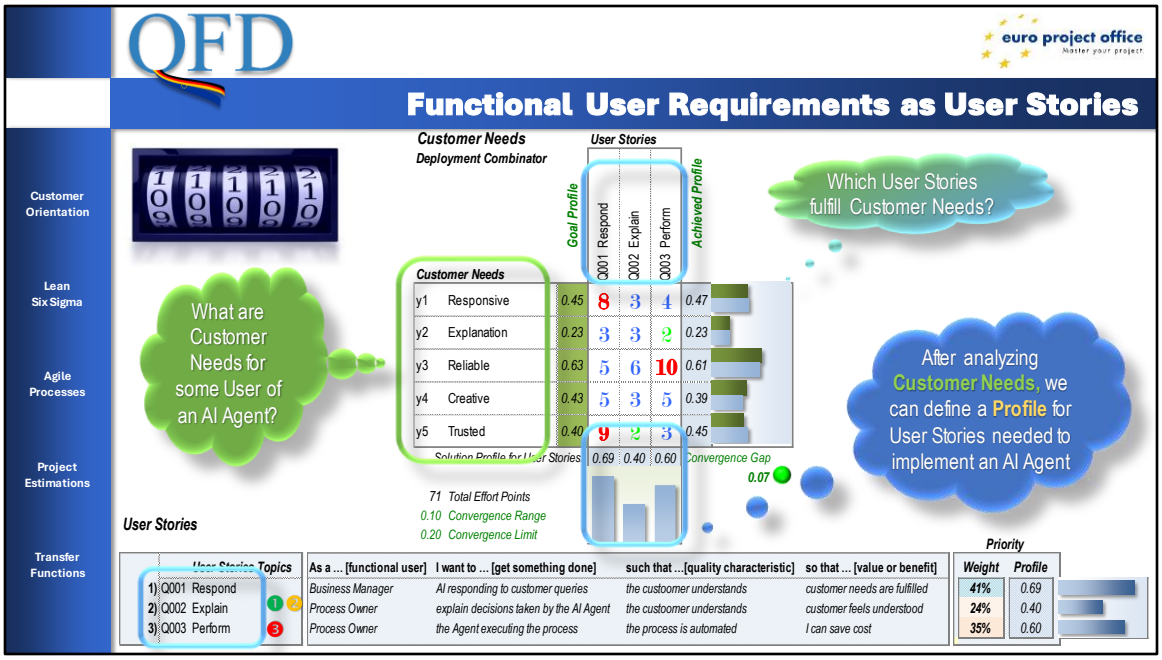
AHP Priorities

Customer Needs	y1 Responsive	y2 Explanation	y3 Reliable	y4 Creative	y5 Trusted	Weight	Ranking	Profile
y1 Responsive	1	3	1/3	1/6	6	21%	2	0.45
y2 Explanation	1/3	1	1/3	1/3	3	11%	5	0.23
y3 Reliable	3	3	1	6	1	29%	1	0.63
y4 Creative	6	3	1/6	1	1/6	20%	3	0.43
y5 Trusted	1/6	1/3	1	6	1	19%	4	0.40

38

Assume, we want an AI Agent that responsibly and sustainably quotes customers on our products and services.

We omit all the escalation stuff that is needed in case the AI Agent cannot issue a viable and approvable quote, for the sake of simplicity.

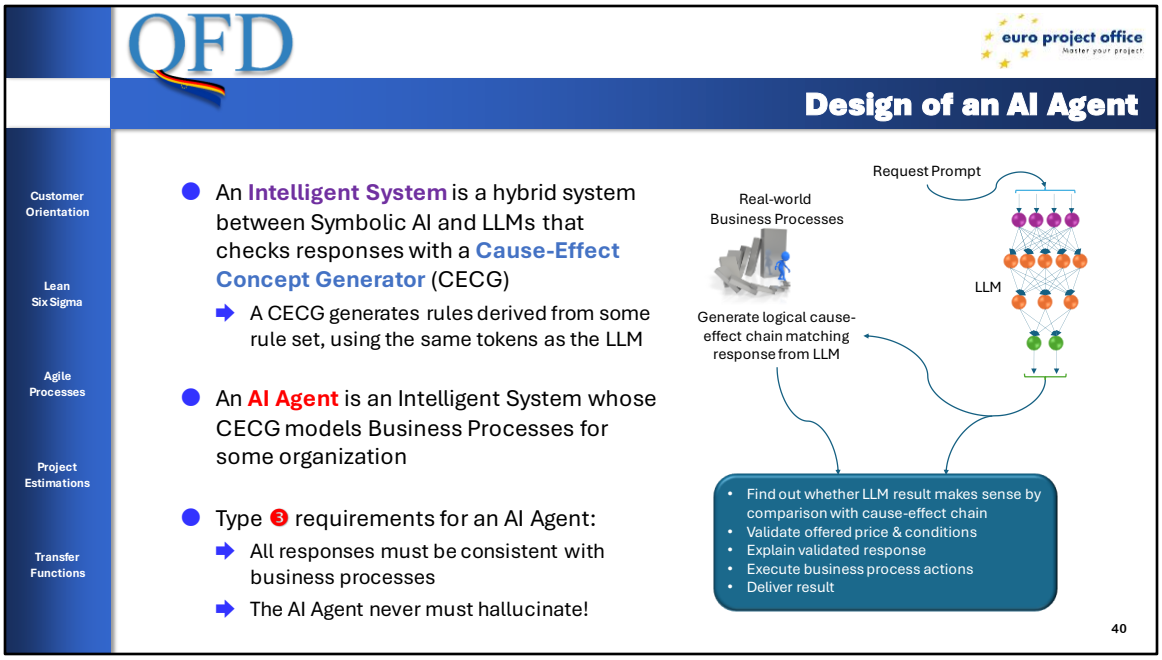


Three User Stories – the first two according to type ❶ and ❷, the third one of type ❸.

The three user stories (in blue) implement a system that meets five customer needs (in green). We need a profile for the priorities of those user stories, which are rather epics, due to the granularity chosen for our Café sample.

The numbers in the matrix represent the number of data movements that contribute to fulfilling customer needs in user stories.

Agile Teams have multiple ways of prioritizing user stories with different methods.

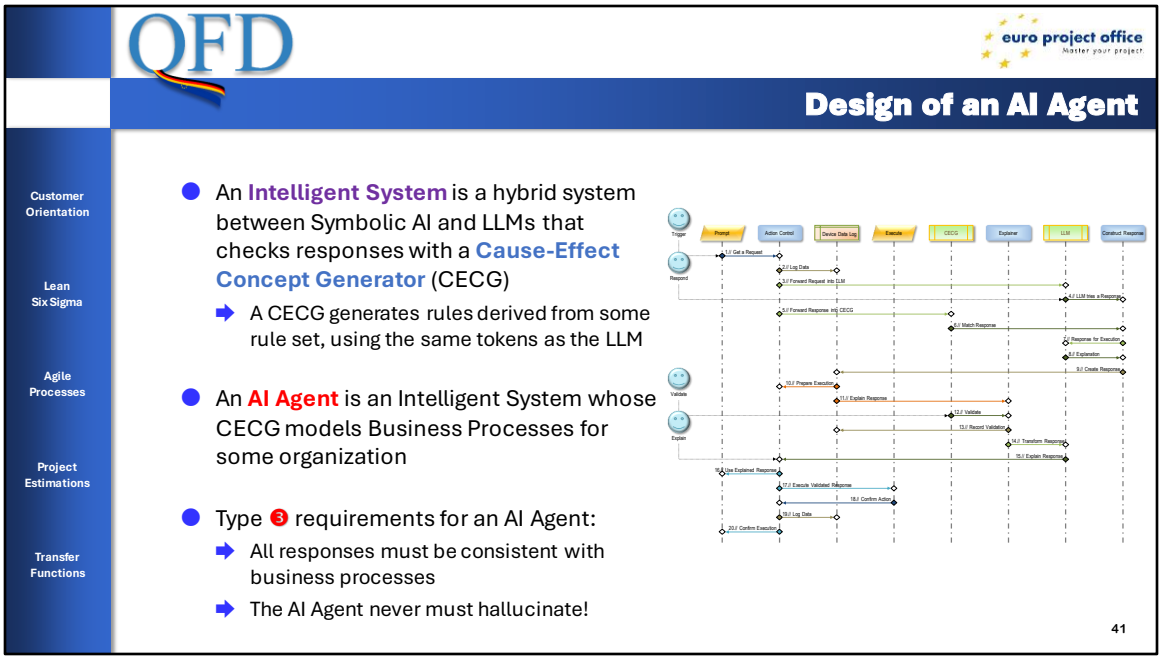


A very basic AI Agent is an intelligent system that can respond to user requests for instance with reference to some real-world domain, e.g., an eShop.

The AI Agent must understand what the user needs using basic knowledge involving cause and effect in his domain – for instance, if you want screws, you likely need a screwdriver. Or if you want sound-absorbing ceiling elements, whether you need them fireproof or not.

Moreover, this AI Agent never must quote wrong prices!

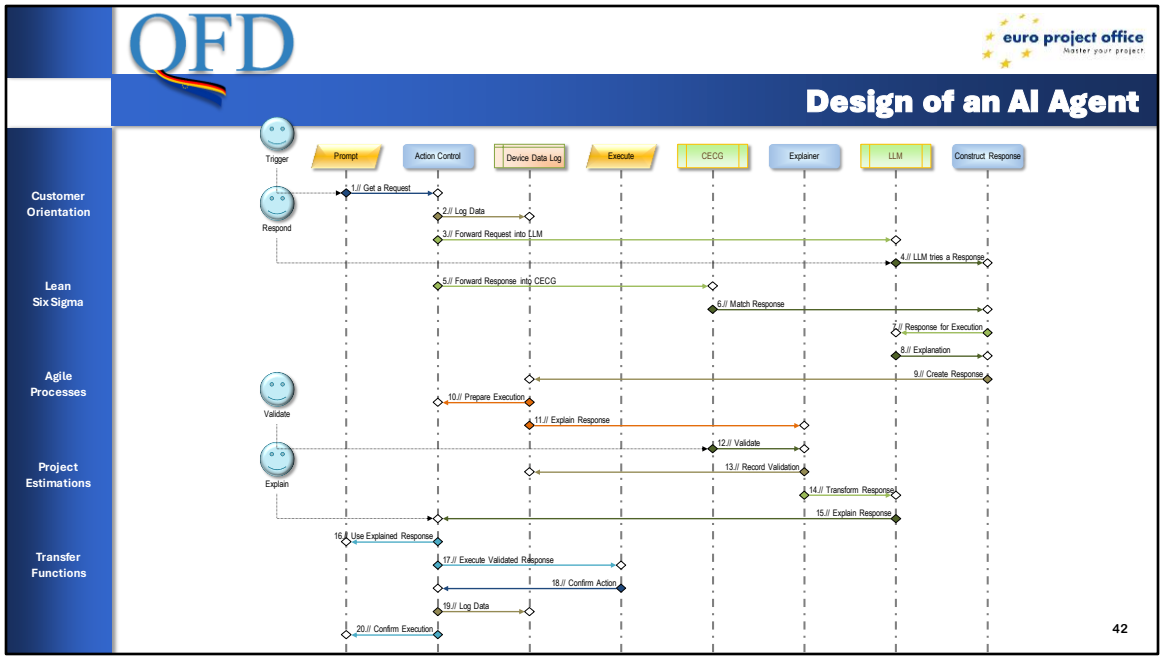
That we must test; otherwise, we run into liability risks.



An AI Agent is best designed by modelling data movements according to ISO 19761 COSMIC because data movements carry the data groups that represent knowledge at different levels of reliability in AI systems.

The above data movement map shows a very simple implementation of the design sketch before. Because only the essential functionality is shown, it fits into this presentation. A real implementation is somewhat more complicated but usually one should set a limit by less than hundred data movements. Thus, a level of granularity is reached that still can be convened to humans, e.g., management and peers.

Obviously, what's below the level of granularity chosen, is not seen by testers and thus must remain untested. This allows testing AI despite the very high complexity an AI Agent or AI engine can achieve. This is not a weakness of the approach, in contrary.it makes testing feasible even with limited resources. Would we care to test the tensor multiplication algorithm that implement a perceptron?



This is how the data movements look like for such a simple AI Agent. It looks well testable; however, we must consider that responses from an LLM never are the same. They are similar but not identical. These requirements are of type ❶ or ❷.

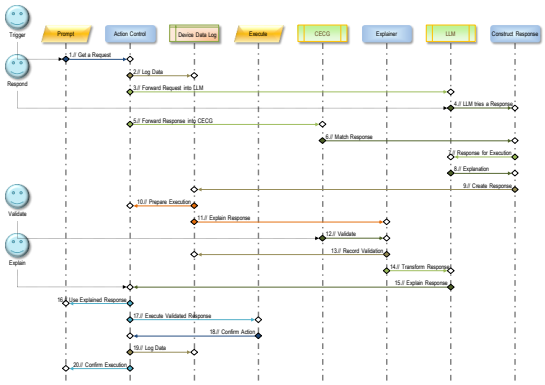
QFD

euro project office
Master your project

Splitting a Data Movement Map into Independent Test Cases

Customer Orientation
Lean Six Sigma
Agile Processes
Project Estimations
Transfer Functions

- Data groups play a major role
  - Collect the referenced data
  - Data groups hold test data
    - You might freely choose test data within data group boundaries
    - Thus, tests become executable as soon as the data groups are known for each test case
- For a test case, it is sufficient to consider the data group as test data for each triggered entry data movement
  - These are the initial graph entries!

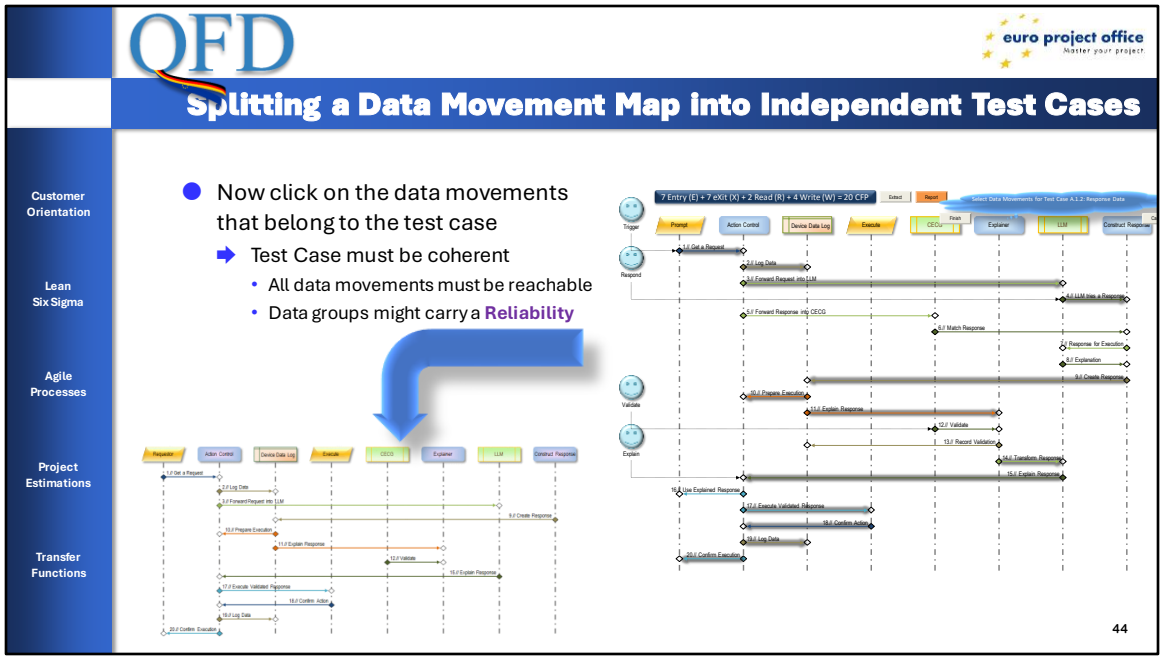


43

The ***Data Movement Model*** contains only those elements that are essential for these three requirements for the AI Agent.

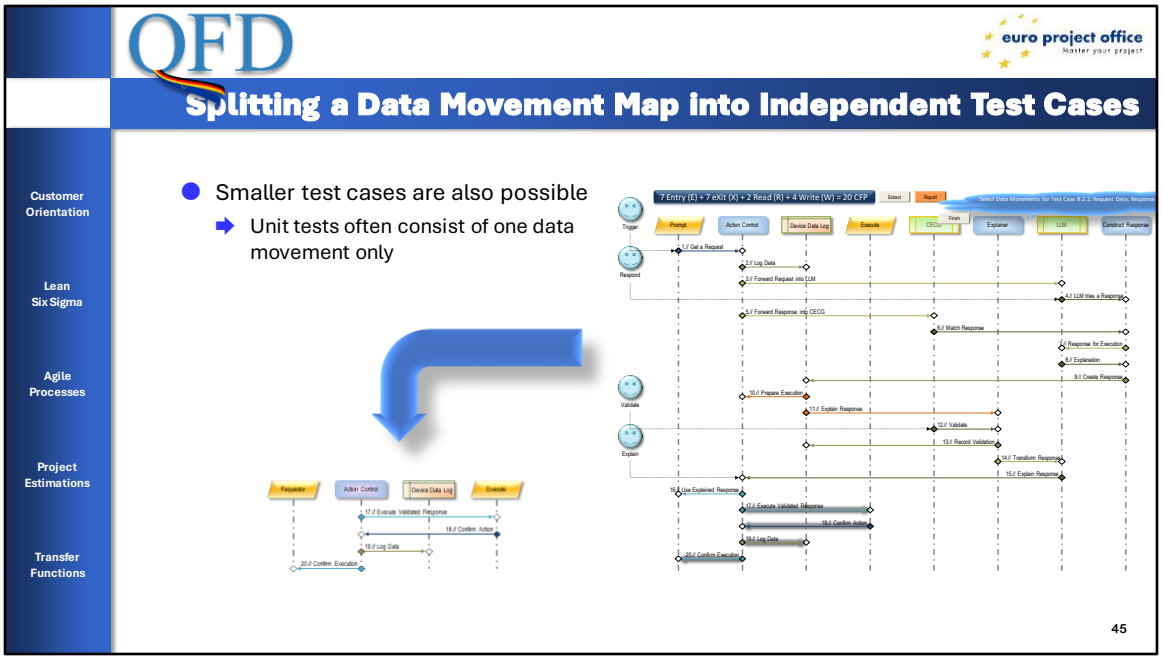
The ***Objects of Interest*** are on a high level, representing a granularity that is sufficient for assessing safety and security.

We expect reliability within some limits – not shown here – but request absolute adherence to Type ❸ requirements. Otherwise, the test fails.

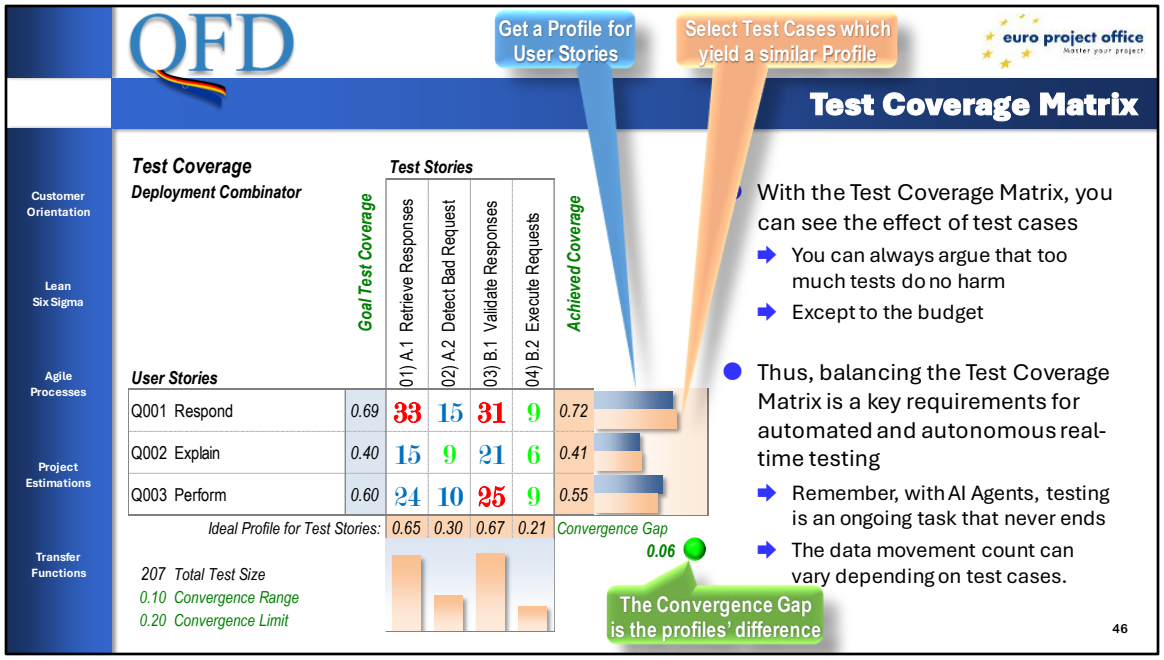


A test case consists of a selection of the data movements that are coherent, i.e., reachable. The data groups for our graph representation (arrow terms) are those at the selection of test data and the one data group that holds the test result.

Each of the data movements might hold not only a data group but a reliability as well, when originating from some AI engine such as an LLM or a Visual Recognition System.



Unit test cases typically consist of one data movement only. However, many tests are small and, for execution, require the ability to fill their initial data groups with actual, sample, data.



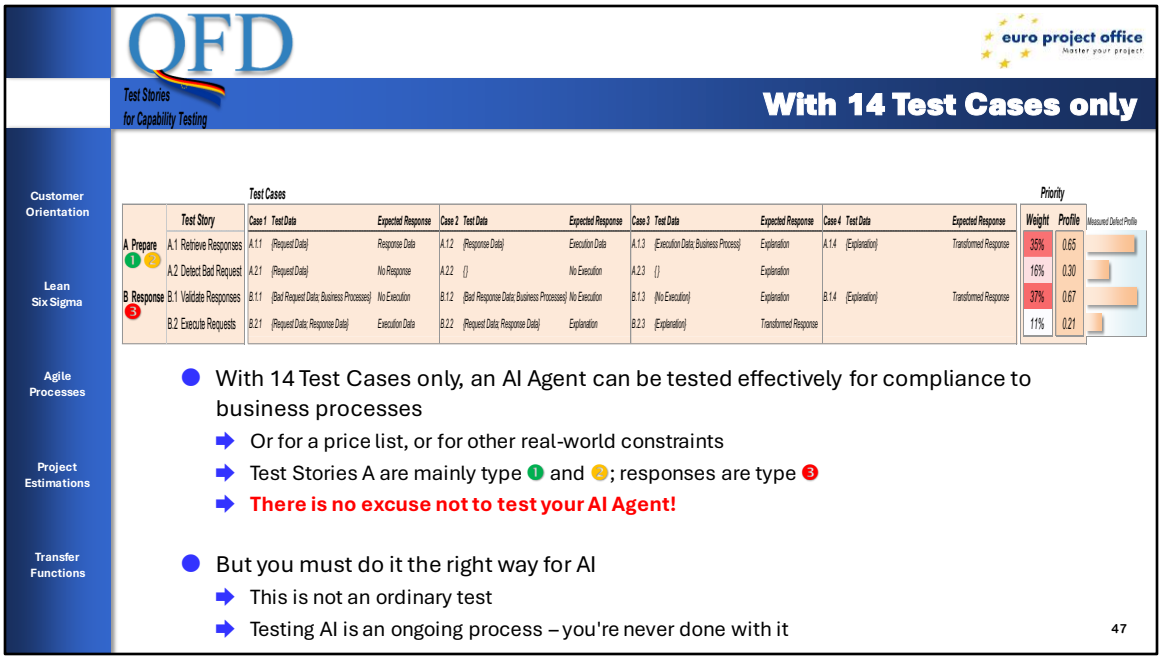
The Test Cases we organize in Test Stories, bringing together what belongs together, in terms of business requirements (typically ***Non-Functional Requirements***, NFR). Then we count the data movements of all test cases that belong to a certain User Story and Test Story. This count yields numbers in the Test Coverage Matrix.

The ***Test Coverage Matrix*** is very dynamic, as both the user story profile and the test cases in use for some test story can vary.

You can add more test cases, but the test coverage matrix should remain balanced, i.e., the ***Convergence Gap*** should close.

The (blue) profile for the user stories is well-known to any agile team, or you can look at my presentation last year in Thessaloniki how to assess customer needs and derive a profile for user stories

In any case, the test coverage matrix allows to sieve test cases for relevance to the user stories. The first two user stories are NFR according to type ❶ and ❷, the third one is a FUR of type ❸.

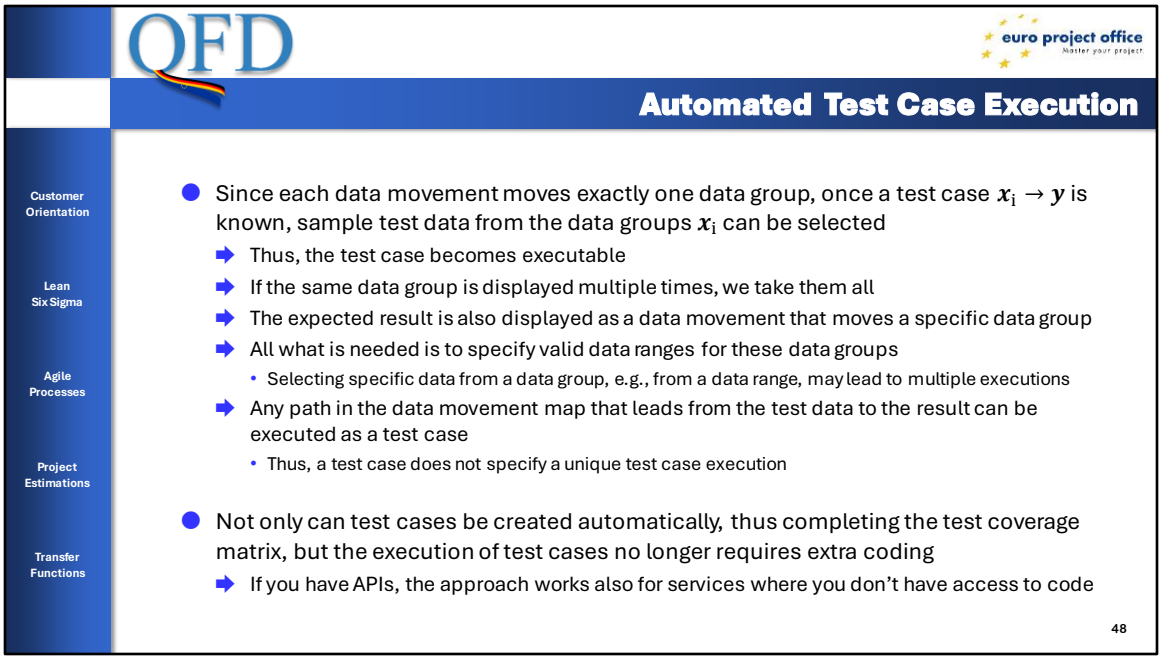


The good news: With 14 test cases only, you can guarantee that your AI Agent is not doing any harm to your organization.

There is no excuse not to test your AI Agent!

The preparatory test stories contains typically type 1 and 2 requirements, the response test stories are of type 3 .

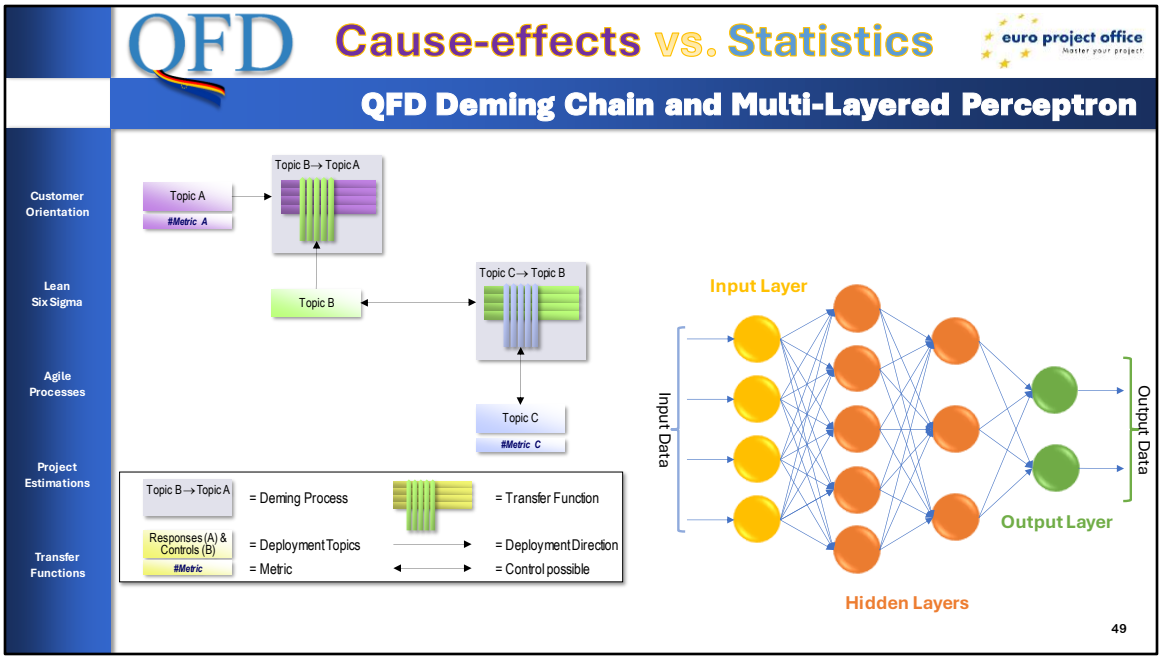
The bad news: when the AI Agent changes behavior, the test cases vary as well, especially with type 1 and 2 requirements. Type 3 results must always conform to defined business processes.



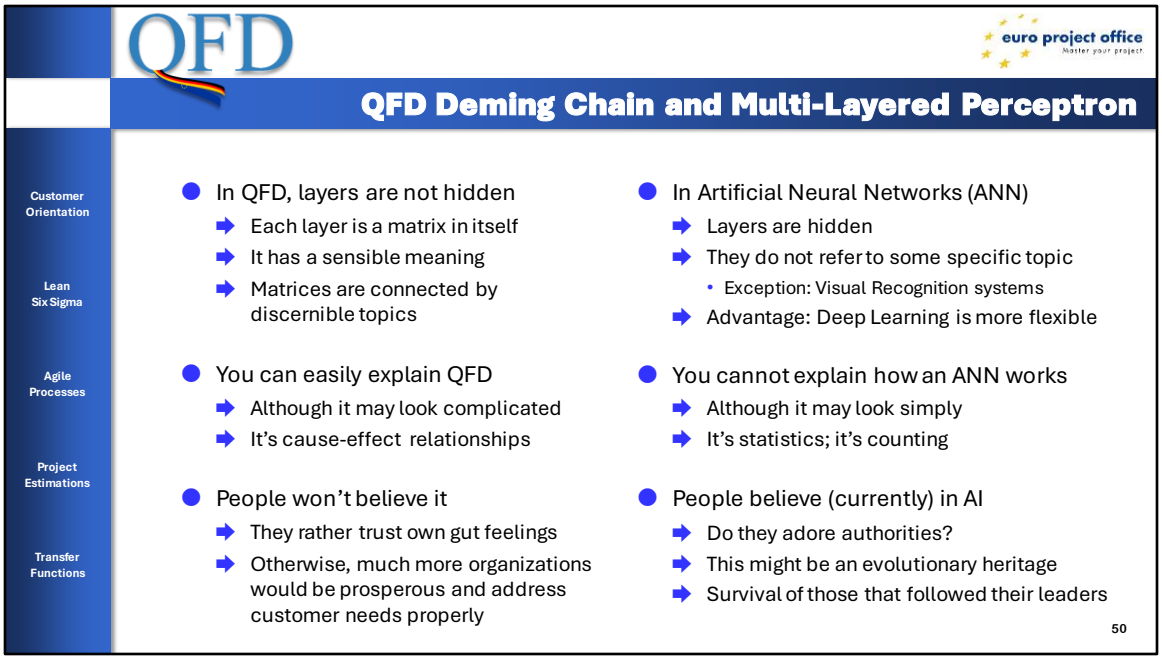
While testing an AI engine is difficult, because it changes behavior, it is always possible to test a functional model.

Nevertheless, it is possible for testing a functional model.

Moreover, if no code is available, e.g., for services, a ***Digital Twin*** might allow to execute test cases involving such functionalities that depend partly or in full on external services, or that need hardware in the loop. In this case, for the Digital Twins, you must provide code for such external services or hardware.



The big difference between QFD and an *Artificial Neural Network* (ANN), most often designed as a Multi-layered perceptron, is that layers in an ANN are hidden. Otherwise, each layer corresponds to a very large, sparse, matrix. The values in the matrix describe, as in QFD, the strength of coupling the input with the output. If multiple entries in a column exist, they strengthen the response. Most matrix cells remain empty, not stimulating any response.

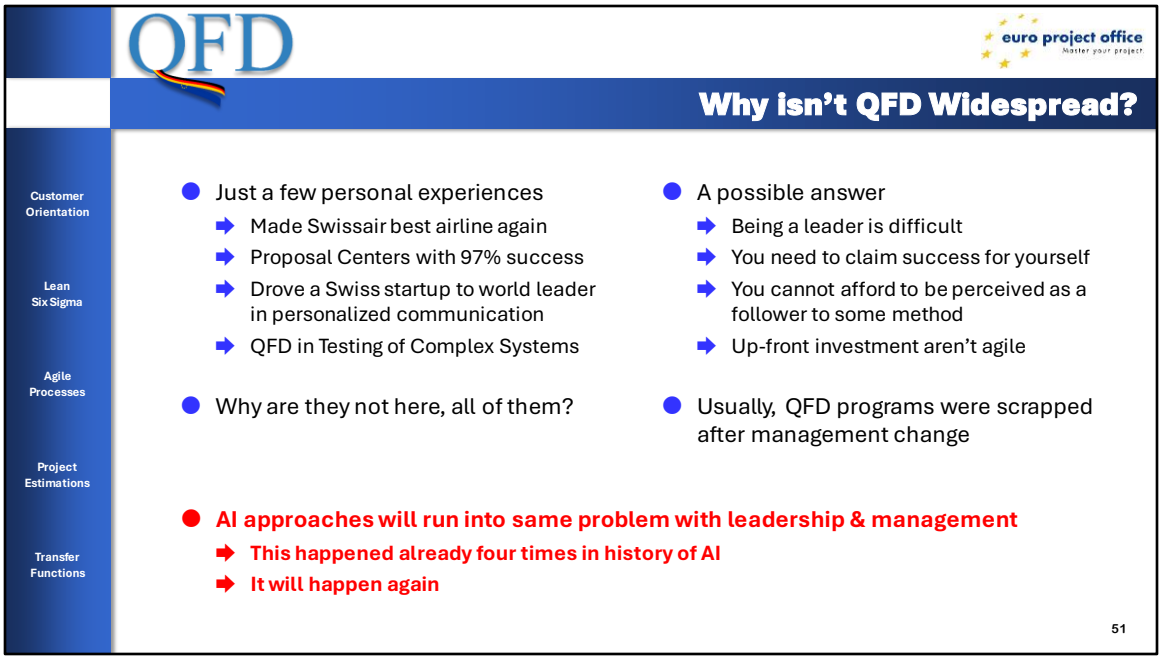


This is a comparison between QFD and AI.

In QFD, the flow of inception is easily traceable; however, it is based on measurements or expert evaluations. Both are not easily comprehensible.

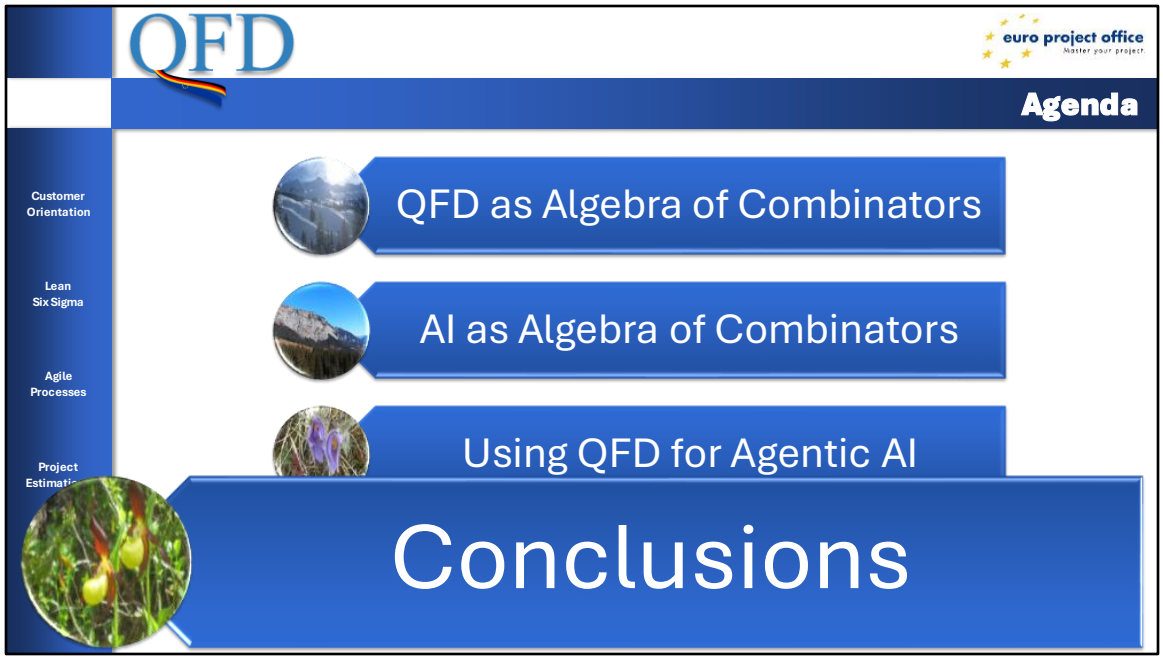
AI has no such problems because the layers are hidden, and thus not comprehensible at all.

However, there has been some progress in AI in terms of skills traceability.

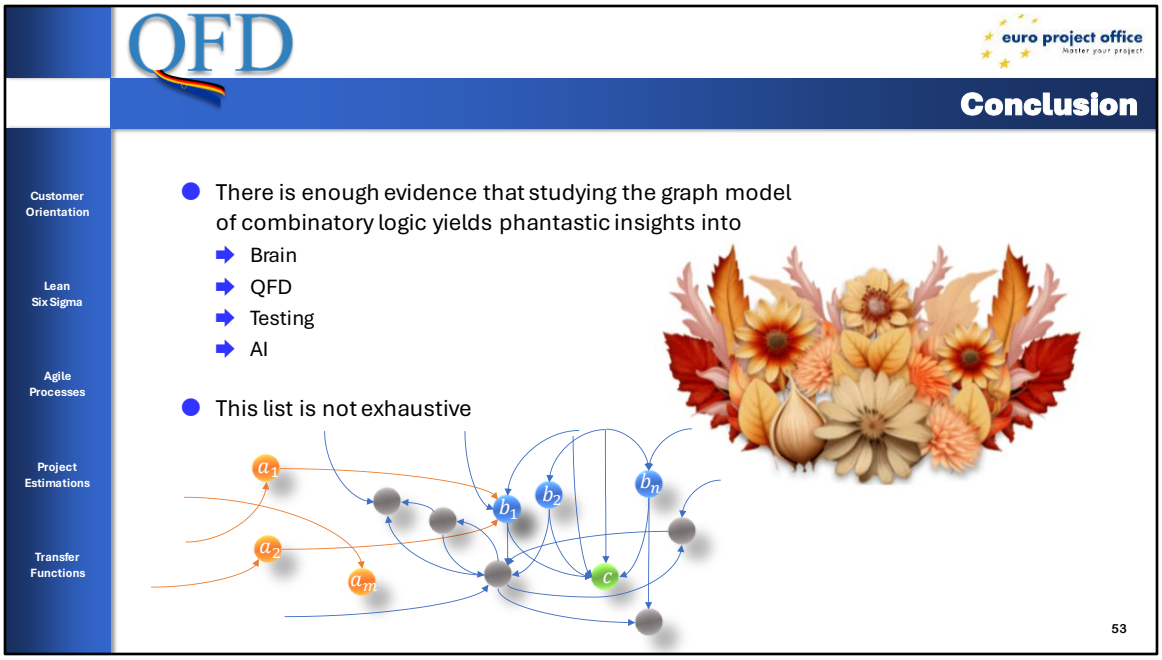


Usually, bad leadership successfully kills QFD, and will kill AI, simply because leaders fear the loss of power and influence.

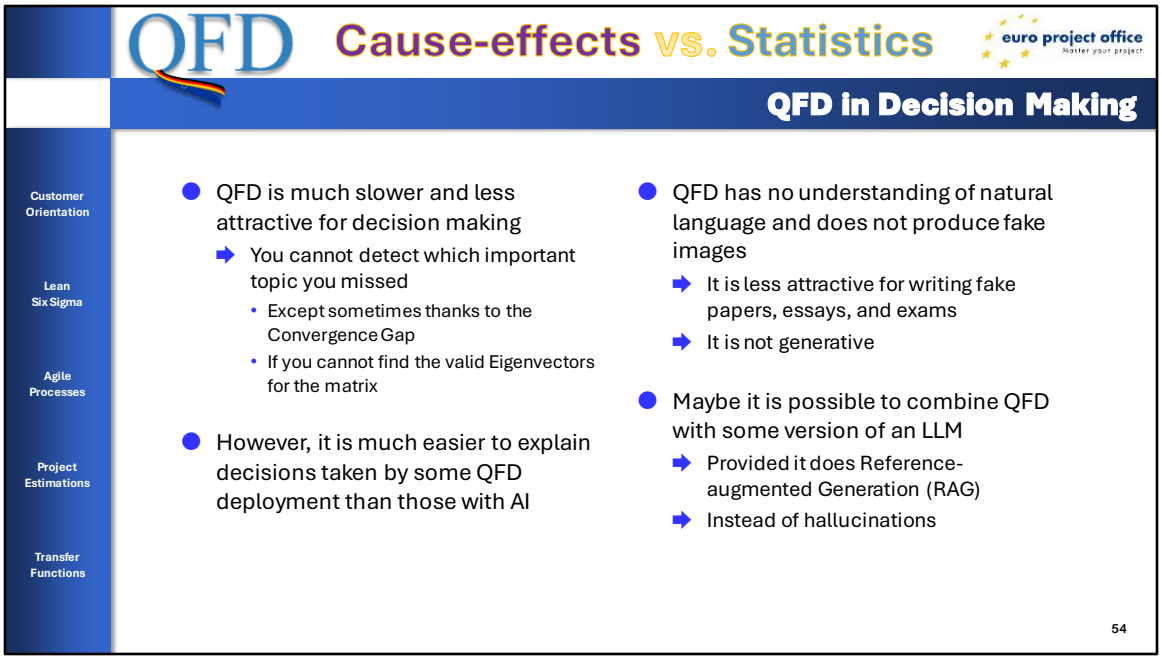
Luckily, some few organizations, typically start-ups, take over and use new methods successfully. It all depends on the stature of leaders.



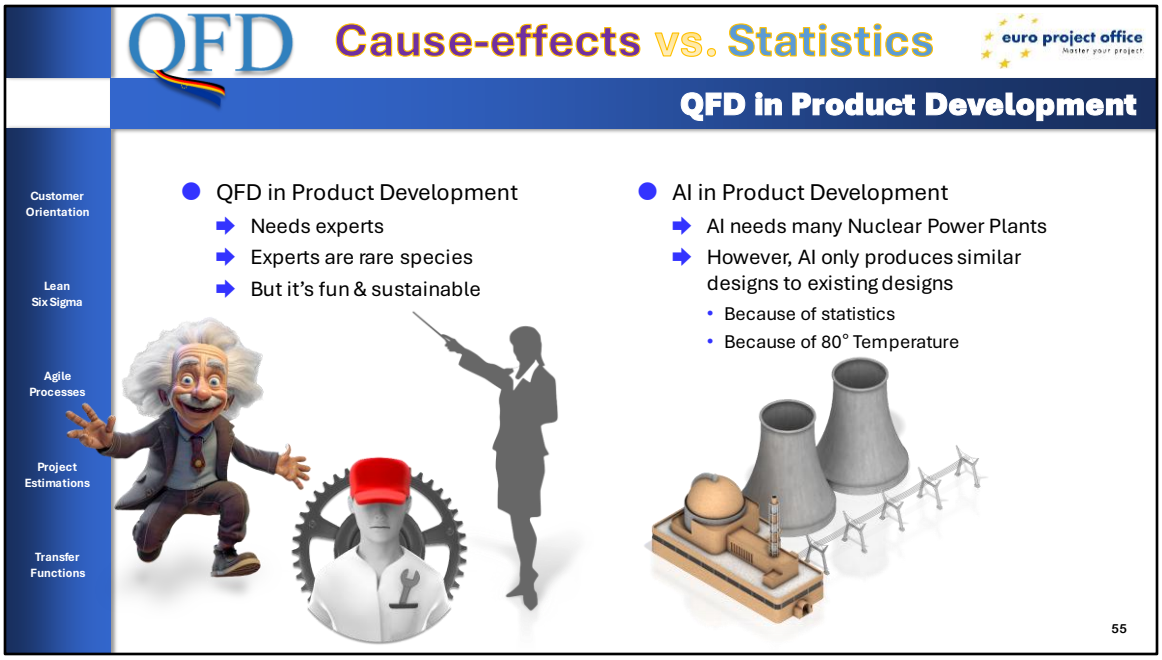
- QFD as Algebra of Combinators
- AI as Algebra of Combinators
- Using QFD for Agentic AI
- Conclusions



The graph model of combinatory logic is filling the gap between logic and nature. It is highly recursive and applicable to any kind of knowledge – QFD, Testing. AI, …



QFD is slow and expensive for decision making, that should go fast. QFD is not good at considering other influential topics not seen by experts. AI simply has higher capacity and considers everything – including the famous glitch when an AI that was trained to distinguish wolves from dogs looked at pictures whether there was snow. All wolve pictures in the training set incidentally were taken in winter with snow.



While I have no big hopes for QFD in decision making, it has big advantages in product development. QFD-designed products easily gain market share and are successful, innovative, and attractive, while AI produces always more of the same. Statistics do not generate new designs, even when guided by customer needs. Customers usually want more of the existing for less money; progress is less important.

Nor is sustainability.

One point to mention in favor of AI: Sometimes, gut feeling is better than reasoning (Gigerzer 2007). ANNs must rely exclusively on their gut feeling, because the biological basis of gut feeling is learning from experience, just like ANNs do.

The speaker has published quite a bit on the subject together with Eberhard Kranich in Duisburg – e.g., in QFD symposia, at SW metrics conferences like IWSM / Mensura; in quality management and testing conferences, also Lean Six Sigma Conference in Glasgow, Strathclyde and Zurich and in the ATINER series of scientific publications.

Managing Complexity appeared 2016 in Logos Press, Berlin: https://www.logos-verlag.com/cgi-bin/buch?isbn=4406

Autonomous Real-time Testing is available since January 2020 with the same publisher: https://www.logos-verlag.com/cgi-bin/buch?isbn=5038