Federico Badaloni

Function-Driven Design Method

A Relational Framework for Complex Systems and AI-ready Information Architecture

Abstract

Function-driven Design, the subject of this paper, is a method grounded in the idea that it is more effective to design a "system" – that is, an app, a website, or a pervasive service – starting from the relationships between its parts rather than from the parts considered in isolation. These relationships can be interpreted as functions, that is, specific ways of addressing a need through a solution. The strength of the method lies in using this concept as a driving element across all phases of the work: to define ontologies, taxonomies, and the logics of content production, aggregation, and processing, up to establishing the product's visible layer. The paper shows how the method's characteristics – semantic clarity, hierarchical structure of statements, and scale invariance – make it particularly suitable for collaborating with generative artificial intelligence systems, both by supporting the overall review process at every project phase and by accelerating the production of deliverables. The method was developed over years of work designing products and services that were often very different from one another, and it was formalized in response to requests for guidelines from the students I have had the good fortune to encounter in my teaching activity.

Introduction

If we think about the distinctive characteristics of the products and services we have seen emerge over the past ten years, we may recall – for example – the increasing continuity between physical and digital realms, the multiplicity and variety of touchpoints, the speed of change in enabling technologies, and the growing density of information.

In general terms, we can observe that value (including economic value) almost always lies in the ability of products to enable, build, or enhance relationships. The specific identity that makes one product distinguishable from another lies in the particular way it does so.

Consider, for example, dating apps, ride-sharing apps, or platforms that allow us to buy and sell goods. The reason we choose one rather than another does not lie in the judgment we assign to the individual entities these apps contain, considered separately, but in the specific relational logic through which these entities are orchestrated, governed, and synthesized into a coherent service.

Within the system, these entities – including ourselves, through our user profiles – continuously form and in-form one another, generating new meaningful events (or relationships, or connections, or processes, if preferred).

This is a phenomenon that was identified early on, fifteen years ago, within information architecture by Andrea Resmini and Luca Rosati when they wrote that "information spaces are processes" (2011), and by Andrew Hinton in his foundational work, Understanding Context, when he stated that "context" is "an agent's understanding of the relationships between the elements of the agent's environment" (2014, p. 25). Contemporary philosophical reflection has likewise described the past decade as a transition "from a web of things to a network of processes and relationships" (Floridi, 2016) [1].

As designers, we not only contribute to shaping the environments already around us, but we also construct new ones, constantly mixing atoms and bits. Our intentions, as well as the techniques we use, profoundly influence the final outcome.

Awareness of the growing impact of our work on people and societies has led us to question, with increasing insistence, the criteria that make our artifacts ethically sustainable. The particularly valuable work of Batya Friedman has shown how central it is to explicitly document human values within the design process.

However, the growing penetration of artificial intelligence into the tools we use (CMSs, design and prototyping tools, ethnographic research tools, etc.) has meant that the increasing diffusion of these systems has also been accompanied by a progressive erosion of the ability to traceâ€"and thus to make understandableâ€"the criteria and processes through which the outcomes of digital or phygital products and services are produced.

As Andy Fitzgerald has pointed out, teams themselves increasingly tend to "focus on the arrangement of resources and functionality, but ignore the methods of arrangement that generate that final result" (2020). This is the perfect recipe for disaster, because "when we no longer even understand how the pieces of the system fit together, we shouldn't be surprised to find (actu-

ally) unintended effects on the rise." [2]

A paradox thus emerges: precisely while we seek greater ethical responsibility, we adopt tools that tend to hide design causality under the carpet of generative automation.

It is within this methodological gap that function-driven design finds its place. If the problem of AI lies in its often arbitrary nature or in its being a "black box," the response cannot obviously be the rejection of the tool, but rather the definition of a more rigorous structure of dialogue with it.

Designing through functions ultimately means transforming the creative process into an architecture of declared intentions, where each AI output is no longer an isolated and inscrutable event, but the manifestation of a predefined semantic relationship and, therefore, one that is documented and governable.

In function-driven design, the function is not a simple static link between entities, but the active form of the relationship. We can imagine it as a vector of intent: a force that defines not only the connection between two points in the system, but also the direction and purpose of that contact. Defining relationships in functional terms means providing artificial intelligence (and the humans collaborating with it) not only with a map of "things" (what they are), but with the grammar of their "whys" (why they interact).

This transforms ontology from a passive description into an explicit decision-making architecture, enabling logical validation of the output.

In a classical ontological model, imagining we are designing a healthcare service, a relationship might be described statically as: "the Doctor owns an Agenda." This is a possession relationship that defines a data geometry, but not an intention. In function-driven design, the relationship becomes a function when it is stated with a vector of intent: "The system shows the Doctor's time availability to Patients who need a visit".

It is in this gap that the difference from simple database mapping resides: by asserting that the function is the active form of the relationship, we perform a semantic leap that takes us from the geometry of connections to the kinematics of system actions.

This is the crucial point for collaboration with AI. Language-based generative models (LLMs) can produce plausible explanations, but without explicit constraints they do not guarantee traceable causal alignment between architecture, behavior, and value or ethics. If we provide AI with mere structural adjacency – for example, the fact that the "Patient" entity is linked to the

"Medical Record" entity – the AI must "guess" the meaning of that link.

If, instead, we define the relationship as an active function, we provide causal logic: "The system validates the Patient's identity before authorizing access to the medical record". In this way, we are not merely describing an architecture; we are issuing an explicit ethical and functional command. The AI no longer has to "hallucinate" product behavior, because the rationale for its functioning has been provided as a structural parameter.

From this perspective, the function ceases to be a mere technical device and becomes the true connective tissue of the project. It is what defines the ontology, preventing it from remaining an abstract and self-referential theoretical exercise. Acting as a unit of meaning and command, the function ensures that every structural choice is oriented toward a concrete experience.

We can thus summarize this logical hierarchy in a formula that guides the entire method: if ontology tells us who we are (Entities), and the phenomenon is how we appear (UI), the function is the engine that transforms being into appearing.

Functions, ontology, phenomena: designing from the inside out

As suggested by Herrera Batista (2020), the reality of design is intrinsically relational. Function-driven design shifts attention from the aspect of the product to its ontological structure, ensuring that relationships between parts are the core of the design process.

"The designed object does not exist as an isolated entity; its ontological reality is determined by the network of relationships it establishes with the user and the designer within a specific context. Its essence is not in its material form, but in its relational function."

If it is from relationships that we extract value, it makes sense to ask whether it is advantageous to adopt a relations – first working method – that is, a method that infers from relationships the formal characteristics of both invisible entities at the ontological level and visible, audible, and tangible entities that determine the appearance of our products.

In my teaching experience, however, I have encountered many students and professionals who took for granted the opposite procedure: to design, we must first visually imagine the result we want to achieve, then imagine visible, audible, and tangible "things." Then, in a kind of backward movement,

we organize them, establish what characteristics these things must have at the ontological level, and finally determine what relationships they must weave among themselves to produce the desired results. In short, a design process that goes from "outside" to "inside" a system.

I also began this way twenty-five years ago. But over time, in my experience as both designer and teacher, I found it far faster, simpler, and more orderly to first define and organize the relationships we want a system to produce, and only once these are clear and exhaustive, to derive from them the formal characteristics of entities and their manifestations. A process from inside to outside.

I should clarify, however, that in this paper I do not intend to use the concept of relations-first as a general thesis about the ultimate nature of reality, but solely as an operational design principle.

The general thesis – which I find extremely interesting, and which emerges in various ways both in the philosophy of information and in quantum physics – holds that reality itself is generated through the set of relationships, processes, and events that traverse it and make it recognizable.

The methodological thesis, by contrast, is more circumscribed and verifiable: in design contexts characterized by informational and systemic complexity, and high touchpoint density (Jones & Kijima 2018; Resmini & Lindenfalk 2021), it is advantageous to adopt procedures that make relationships explicit and orderable before permanently fixing the form of entities, their organization, and their representation.

The aim of the method is, in fact, to produce outputs that are simple, modular, structured, coherent, and hierarchically ordered, allowing us to collaborate with artificial intelligence in an extremely fast, documented, and effective way. In short, if we want to design with (and also for) AI, we must be able to describe reality in terms of functions that are operable by machines and understandable by humans.

Throughout the previous sections I have repeatedly mentioned the centrality of the ontological aspect of design. Herrera Batista (2020) emphasizes in particular how this guarantees the coherence of meanings and functions in design:

> *"To understand design beyond its appearance, research must penetrate the deep structure of its ontological reality. If design remains only at the level of aesthetics, it fails to fulfill its role as a generator of meaning and functional coherence."*

A review of course syllabi and bibliographies in our field, however, reveals

that far more effort has been devoted to the research phase (user needs, stakeholder objectives, constraint analysis, etc.) and to defining the appearance of products and services (interfaces, interactions, etc.) than to ontology.

Yet ontology is precisely the domain in which possible relationships within a system are defined and enabled. It is true: this is the most abstract aspect, and therefore tends to be the most difficult to address, especially for beginners, but it is truly the "matrix" of every artifact. The word "matrix" derives from the same Indo-European root as the word "mother," indicating the capacity to generate.

Although it may be metaphorically described as the DNA of the project, ontology constitutes – citing Gruber's classic definition (1993) – an "explicit specification of a conceptualization." In this sense, ontology in function-driven design is not a passive description, but the formalization of the system's internal logic that allows meaning to be grounded unambiguously.

Precisely because of this characteristic, ontology today more than ever is the ideal foundation on which to build collaboration with artificial intelligence in the design process. As Seale (2025) observes, "ontologies provide the formal structure to anchor meaning and harmonize wildly different data sources into a coherent semantic layer."

An ontology is not merely a description of a domain, but the formalization of its internal logic: it defines syntax and hierarchy, establishes which entities may exist, how they may relate, and which constraints govern those relationships. Ontologies are therefore a prerequisite for enabling generative models to build, enrich, and maintain interpretable and consistent knowledge graphs.

Even when ontology is included in curricula and design processes, little attention is usually paid to the method required to generate it. With rare exceptions, [3] it is expected to take shape spontaneously in the designer's mind at some point along the way. And what if that does not happen? What if multiple hypotheses form in our minds, all of which appear equally valid?

As observed by Polanyi (1966), much expert knowledge resides in a "tacit dimension" that is difficult to communicate verbally. Function-driven design aims to formalize this competence by translating the designer's intuition into a structure of explicit relationships.

This act of encoding is the necessary step not only to make the project operable by artificial intelligence, but also to transform subjective experience into a replicable teaching method that can be shared with students and collabora-

tors.

Allow me to conclude this introduction with a note clarifying the aims of this paper: I will not undertake a systematic comparison with other established approaches to product and service design. Here I limit myself to clarifying the conceptual and operational scope of the function-driven design method, making explicit its assumptions, internal logic, and the conditions under which it proves particularly effective.

The aim is not to replace existing frameworks, but to integrate into them a logical engine capable of governing semantic complexity while at the same time generating a rigorous documentary trace of design decisions. By making rationales and operating principles explicit, the method ensures that every function is not merely an effective technical solution, but a transparent, verifiable decision consistent with the project's value architecture.

At this point, it is essential to clarify how the notion of â€˜function' in this framework departs from its traditional use in classical Functional Requirements Specification (Sommerville, 2016; ISO/IEC/IEEE 29148:2018). In standard software engineering, functions are typically treated as behavioral constraints defined after conceptual architecture.

In contrast, within this method, the function acts as a primary generative act and an epistemic operator. It is a scale-invariant linguistic unit where the relationship (the verb) actively determines the emergence of entities (the nouns), thus generating the system's ontology rather than merely documenting its behavior.

By shifting from a descriptive-static list to a relational-generative grammar, the method provides a high-density semantic structure. This not only ensures logical verifiability and traceability of design intentions but also creates a 'machine-readable' logic that drastically reduces ambiguity when collaborating with generative artificial intelligence systems.

Beyond narration: the system as subject

The growing pervasiveness of information and the density of touchpoints have made design models based purely on user narration (such as User Stories) intrinsically fragile. In highly complex contexts, the attempt to map every usage scenario from the human actor's point of view often leads to excessive fragmentation: disproportionate effort is spent describing atomic use cases, while losing the ability to govern overall coherence and the traceability of decisions.

The conceptual shift of Function-driven design consists in moving the focus from what the user does in the system to what the system does for the user.

By reformulating the unit of measure of the project, a logical regularity emerges: every intent can be expressed through a statement in which the "system" is the constant subject and the action is directed toward a recipient. In this context, the term "function" takes on a meaning close to that used in mathematics: it acts as an operator capable of mapping an element of the set of needs (user and stakeholder objectives) onto an element of the set of possible solutions.

The product is thus conceived as a complex graph in which functions are not mere textual descriptions, but edges – or vectors of intent – that connect "need-nodes" to "solution-nodes." This graph is not flat, but articulated into functional layers characterized by different degrees of abstraction and stability: deep layers, that host the most abstract and invariant functions (e.g., in a healthcare platform: "The system connects the patient's need for care with the doctor's time availability"); and surface layers, that describe punctual and visible (or tangible, or audible) behavior (e.g., "The system shows the Patient the list of doctors, their specializations, and the first available date").

This hierarchical stratification reflects Stewart Brand's principle of "pace layering" (2000), the idea that different layers of a complex system change at different rates- where the deeper, more stable layers provide the foundation for more frequent changes at the surface. By anchoring the system's ontology to these invariant deep functions, the method ensures that the core architecture remains resilient even as touchpoints and interfaces evolve.

In this transition, the function acts as the engine that transforms being (ontology) into appearing (phenomenon). This hierarchical organization transforms ontology from a passive list of entities into a dynamic structure: entities are no longer predefined data, but progressively emerge as effects of modeling these functional edges. It is on this granular yet hierarchical logical stability that the operational procedure of the method is based, and on its ability to provide artificial intelligence with explicit causal logic rather than mere structural adjacency.

This logical stability is not only a conceptual requirement, but also a response to a recurring operational failure in design workflows: the rigid separation between ideation and execution. Such a fracture often produces two side effects: (1) brilliant ideas that are hard to implement; (2) implementable solutions that are poorly aligned with needs and objectives. The Function-driven Design Method is intended to reduce this gap by forcing all disciplines – from those who govern business objectives to those who implement soft-

ware – to discuss the system through the same format: simple, verifiable, and hierarchically organized functional statements. This ensures that feasibility is verifiable and that the hierarchy preserves the traceability of decisions throughout the process.

The main function

As a first step, the team takes as input the outcomes of research and as-is analysis: user needs, requirements of those who will produce and maintain the system, technological constraints, timelines, budgets, and available skills. This set of evidence is not yet "design": it becomes design when it is translated into an ordered set of functions.

The first design act is the collective definition of the main function. It is a sentence that declares what the system does to generate value in the broadest possible form.

It establishes the system's value promise, delimits the field of alternatives, and provides a shared criterion for prioritizing what follows. Precisely because the main function is a high-density act of synthesis, it is advisable for it to be defined by a multidisciplinary team. It may take time to reach shared understanding, but that time is "design work": it eliminates ambiguity, makes priorities explicit, and allows the team to verify that it is truly aligned on objectives.

In the case of our healthcare platform, the main function might be: "The system connects the patient's need for care with the doctor's time availability and specialization."

The advantages of controlled language

A shared language is the key. In 'Information Architecture for the World Wide Web and Beyond,' Morville, Rosenfeld, and Arango emphasize that coherence among instances of an architecture depends on consistent language use and on the order and relationships among its elements (2015). In the Function-driven Design Method, this idea is transformed into an operational rule: functions are expressed using controlled, uniform, and unambiguous language, so that they can be discussed, validated, and reused throughout the entire process. Arango (2020) observes that

> *"(w)e're concerned with frameworks of language and distinctions and relationships, but also with the underlying principles that establish and govern them. Ideally, the language structures we define – the relationships and distinctions*

> *they enable – continue to serve their purposes as they evolve and extend over time, sometimes long after the people who started them have moved on. This requires that we know how and why these configurations come into being, and what makes them an ongoing concern. It also requires that we get good at articulating these principles clearly."*

The Function-driven Design Method uses language, rather than as a descriptive tool, as a project infrastructure, and fixes it in a controlled and standardized form. In this way, functional language becomes a true code for expressing ideas.

Using the same logic to generate specifications throughout the entire project offers all team members a guiding thread that runs from user needs to a single interface button, from stakeholder objectives to decisions about a particular relationship among entities, from a constraint to a specific data extraction algorithm.

It is the precondition for enabling everyone to express their talent and creativity at their best, and for allowing and encouraging each person to participate in the discussion even in areas beyond their own.

Just as in code, indentation, syntax, and recursion make a complex structure readable and maintainable, in Function-driven design the hierarchical organization of functions makes the system's internal logic explicit.

To achieve these goals, the method requires that each function be expressed according to an invariant pattern:

- `Subject: "The system"`
- `Verb phrase: the action performed by the system`
- `Direct object: the solution/resource through which the action is carried out`
- `Recipient: the actor to whom the action is directed (user, operator, external system)`
- `Specifications (when needed): time, mode, conditions, ordering`

The use of the subject "the system" is deliberate: it avoids fixing the artifact's form too early (app, website, phygital service) and keeps open an ecosystemic reading of the product.

Let us return to our example. We said that the main function of our healthcare platform could be: "The system (subject) shows (verb) doctors' time availability (object) to patients (recipient) based on their geographic location (specification)."

Immediately following the alignment on the main function -typically within the same collaborative session to maintain momentum and conceptual consistency- the team identifies the immediately subordinate macro-functions and organizes them into a hierarchical map (functional tree). For example, derived from the healthcare platform's main function, a macro-function for the patient sector could be: "The system allows the Patient to search for Doctors by medical specialization and geographic location".

It is already possible to observe here how the nouns used -such as "Doctor" and "Specialization" -are not merely labels, but the actual precursors of the system's entities. As will be discussed in the following sections, the decision to model "Specialization" as a simple attribute or as a standalone entity depends precisely on whether dedicated functions are assigned to it, such as showing its description or managing its associations.

In this way, the same working method covers both the design of the user-facing layer and what makes the system operable, maintainable, and integrable.

Through successive iterations, the team proceeds from the most abstract and stable levels to more concrete ones, eventually describing the system's "leaves" with the degree of detail needed to guide execution.

Thanks to grammatical coherence and logical subordination, functions are not mere textual statements, but become operational units that can be analyzed, compared, derived, and transformed systematically. In this sense, functional language does not merely describe the system: it constitutes an operable conceptual model. The internal hierarchy is not a mere representation: it is the structure that maintains coherence between strategy and detail.

Defining the ontology

When the functional tree is sufficiently stable, the team moves on to defining the ontology.

The first step consists in listing the concepts that have been mentioned in the sentences used across the different functions. Each concept is then analyzed to determine whether it should be modeled in the system as an entity, an en-

tity attribute, a relationship, or a type of relationship.

A practical rule: if additional attributes or dedicated functions (management, association, updating) are described for a concept, it will necessarily be an entity; if it does not require further articulation, it can be considered a simple attribute of an entity or a particular type of relationship.

The outcome of this phase is a draft entityâ€“relationship model to be validated with those who will implement the database.

From the analyzed functions emerge the concepts of "doctor," "time availability," "specialization," "patient," and "geographic location." Imagining a complete functional tree, we might find functions such as: "the system allows the doctor to enter and modify their name, surname, email, specialization, and time availability." It would thus be evident that "doctor" must be considered an entity with its own attributes, namely name, surname, email address, specialization, and time availability.

If, however, the tree included, for example, another function stating that "the system shows the user a brief description of each medical specialization," we might consider modeling "specialization" as an entity in its own right, and provide that each doctor associates, in the back end, their instance of the doctor-entity to a specific instance of the specialization-entity.

Functional maps

At the end of the work on the tree, the team determines which functions must be performed in dedicated environments. If designing a website or an app, these environments will be pages; if designing a pervasive environment, these environments dedicated to performing a function may also be physical.

The function "the system shows the user a brief description of each medical specialization" could be performed on a dedicated page within which the system carries out subordinate functions such as: "the system shows the specialization title," "the system shows the specialization description," "the system allows the page to be printed," as well as more standard functions such as "the system allows returning to the homepage," and so on.

Our platform might also have pervasive features, for example functions such as "the system allows users who have a valid booking to access the office." In this case, the functional map to be produced could, for example, concern the entrance to the doctor's office, where turnstiles could be installed that open via a QR code received by the user in the booking confirmation email.

In practice, the team draws as many rectangles as there are environments. Inside each one, it inserts an ordered list of subordinate functions needed to perform as effectively as possible the function that must be carried out at that specific touchpoint.

Functional maps are relatively simple and quick to produce, because they are built by reusing – through "copy and paste" – the functions already defined in the functional tree.

They are a kind of "logical debugging" of the design process, because they almost always bring to light, and allow one to fix, gaps, inconsistencies, or ambiguities in the functional tree.

Before the spread of generative tools, the tree and the functional maps constituted the basis from which teams manually produced wireframes and, when needed, prototypes. Today, the same structure can be used as direct input for tools that automatically generate layouts and interfaces, drastically reducing the time needed to move from conceptual modeling to the representation of the system.

It is precisely this continuity – from the definition of functions to the generation of interfaces – that makes the Function-driven Design Method particularly suitable for collaboration with artificial intelligence systems, as will be discussed in the next paragraph.

Collaboration with artificial intelligence

The intrinsically formal, relational, and linguistic nature of the Function-driven Design Method makes its deliverables extraordinarily legible to Large Language Models (LLMs). Functions, defined as relationships between a need and a solution, constitute design units endowed with stable syntax and rich semantics: an ideal environment for AI, which performs best when given structured inputs characterized by internal regularities.

The first crucial element is the invariance of functional language. Because each function strictly follows the pattern "system + verb + object + recipient," AI recognizes the patterns with high reliability. For a language model, such uniformity produces a significant reduction of ambiguity: AI does not need to interpret heterogeneous narrative styles, but can extract, classify, and transform functions with a reduced margin of error.

This property is complemented by the hierarchical organization of the functional tree. This is not merely a graphical representation, but a semantic

structure that enables generative models to: build conceptual maps and identify semantic clustering; extract ontologies by identifying parentâ€“child dependencies; and automatically convert the project into computational formats (such as RDF structures or Knowledge Graphs) used in reasoning systems, with validation pipelines.

Functional maps then act as "intermediate representations" between concept and interface. For AI, these maps are low-fidelity wireframes in which structure emerges from logic rather than from graphics. The model does not have to "guess" the designer's intentions: by working on the functional map, AI is provided with a representation that makes the system's functioning explicit before its visual formalization.

A distinctive advantage is integral derivativity: every element of the system (entities, attributes, behaviors) derives explicitly from a function. For AI, this means it can act as a structural proofreader: it can identify "orphan" elements (not justified by a superordinate function) or logical gaps (lack of correspondence between front-end and back-end functions, inconsistencies between the ontological model and functional maps, and so on).

The method follows a top-down process (from strategy to detail), but allows bottom-up validation: top-down (construction), where the AI helps the designer expand the main function into coherent sub-functions; bottom-up (inspection), where the AI can trace back from the most granular functions ("leaves") up to the main function, verifying that each technical decision is consistent with the initial value promise.

If, in the details of an interface, AI suggested a "social sharing" function for medical data, a bottom-up check would immediately flag its inconsistency with the validation and protection function for the medical record defined in higher-level functions.

Examples of interaction and prompt transformation

The claim that large language models struggle to generate coherent data structures starting from subjective narratives is not a theoretical assumption, but a consequence of so-called semantic noise. In traditional methodologies, such as User Stories, the emphasis is placed on the user's desire and the final benefit obtained; in Function-driven design, the emphasis is placed on operation and structure.

To understand the difference, let us compare how AI processes two different kinds of input referring to the same scenario (booking a medical visit).

Example A: The narrative approach (User Story)

"As a patient, I want to be able to book a visit with my general practitioner in order to receive treatment and feel better." In this case, AI is faced with a narrative that mixes three different layers:

- Intention: "I want to be able to book."
- Social context: "my general practitioner" (which implies an unspecified relationship of ownership or assignment).
- Benefit: "feel better."

Faced with this input, an LLM tasked with generating a database schema tends to extrapolate unnecessary entities or confuse verbs with objects. It might create redundant tables such as "treatments" or "health states," or fail to define the central entity "visit" as the object that emerges from the interaction, precisely because the sentence is focused on the human subject rather than on the system's action.

Example B: The function-driven approach

"The system shows the Doctor's time availability to Patients who need a visit." Here semantic noise is eliminated. The instruction is an explicit decision-making architecture that AI can map instantly onto a formal schema:

- Logical subject: System (the agent performing the action)
- Object: Doctor.
- Object attribute: Time availability (the resource to be extracted).
- Recipient: Patient.
- Recipient condition: Need for a visit (the target and the logical filter).

Whereas in the User Story AI must interpret a will, in the function it only

has to execute a structure. In the first case, AI acts like an interpreter of texts; in the second, it acts like a compiler. The presence of the specification "who needs a visit" provides AI with an explicit filtering parameter, preventing it from "hallucinating" an unrestricted view accessible to anyone and anchoring the design to a stringent logic.

In this sense, functions act as "assembly instructions" that provide AI with solid grounding (semantic anchoring) to avoid, or greatly reduce, the possibility of hallucinations.

Discussion: toward an ecology of relationships

The Function-driven Design Method has been applied and refined over the past decade in numerous professional contexts characterized by high informational and systemic complexity. The method has demonstrated its effectiveness in heterogeneous domains, including large-scale editorial systems and information products with high semantic variability; pervasive digital services that integrate physical and digital components; and models of information governance and optimization of internal editorial workflows.

In these scenarios, Function-driven Design has acted as a catalyst, making it possible to democratize design, accelerate decision-making processes, and ensure ontological coherence. It democratizes design since the simple syntax allows designers, developers, and non-technical stakeholders to discuss the system's structure without linguistic barriers; it accelerates decision-making processes because sharing a common view of the "whys" (functions) before addressing the "hows" (layout) drastically reduces conflicts and approval times; and ensures coherence by facilitating the rapid generation of precise taxonomies and ensures that technical implementation never betrays the initial strategy.

As discussed at the outset, the validity of a method lies in its ability to make knowledge explicit and transmissible. Teaching function-driven design within executive master's programs and master degree programs has confirmed this property: hundreds of students with different backgrounds (humanities, computer science, managerial) have adopted the method in a short time.

The consistency of results achieved in thesis projects and workshops shows that the scale invariance of the process makes it a versatile tool, capable of adapting both to a small application and to an enterprise-level information system.

Across the last ten editions of the university Executive Master's program in

Information Architecture and User Experience Design, which I founded at IULM University in Milan, Italy, students have always been able to design functional trees, ontologies, functional maps, and the main wireframes from scratch within the 20 hours allotted to the final workshop.

In the most recent edition of the program, which ended November 2025, within the same time frame students were also able to produce fully functioning prototypes, thanks to providing functional trees and functional maps as input to artificial intelligence systems.

Personally, over the past year I have been able to estimate a reduction of approximately 40% in project delivery time compared to 2024, thanks to the virtuous combination of function-driven design and AI. This reduction is due not only to faster checks of consistency and completeness in functional trees, and to the increased speed of front-end and back-end interface production, but also to a radical shortening of internal team alignment and stakeholder approval cycles, enabled by the extensive use of prototypes.

The estimate is based on the times recorded in Wrike, the task management tool used within the company, calculated as an average across approximately 10 redesign projects – comparable in terms of budget, team size, and technical complexity – for websites and apps of news outlets, as well as audio and video streaming platforms belonging to the publishing group where I work.

A humanistic vision for the AI era

In conclusion, the Function-driven Design Method proposes a shift in perspective: it moves the emphasis from visible artifacts to the generative elements of the system. Although its structure makes it a powerful ally for artificial intelligence – by providing that "logical engine" which prevents hallucinations and arbitrariness – the method is rooted in a profoundly humanistic view of design.

Information architecture is not understood here as a mere arrangement of data, but as work of weaving among concepts, people, and systems. Designing, ultimately, means taking responsibility for the relationships we build and for the values they convey. We are what we connect.

References

Arango, J. (2020) Elemental, or how information architecture makes us smarter https://jarango.com/2020/02/22/elemental-or-how-information-architecture-makes-us-smarter/.

Atherton, M. and Hane, C. (2017) Designing Connected Content. O'Reilly Media.

Badaloni, F. (2020) Progettazione funzionale.

Barrasa, J. and Webber, J. (2023) Building Knowledge Graphs. O'Reilly Media.

Brand, S. (2000) The Clock of the Long Now: Time and Responsibility. Basic Books.

Fitzgerald, A. (2020) Delivering Information Architecture. https://andyfitzgerald-consulting.com/insights/delivering-ia.

Floridi, L. (2016) Internet of Things. Eforumita CISCO Conference. https://www.youtube.com/watch?v=Jm4DeJ1jng0.

Friedman, B. and Hendry, D. G. (2019) Value Sensitive Design: Shaping Technology with Moral Imagination. The MIT Press.

Gruber, T. R. (1993) A translation approach to portable ontology specification. Knowledge Acquisition, 5(2): 199â€“220. doi: 10.1006/knac.1993.1008.

Herrera Batista, M. A. (2020) The Ontology of Design Research. Routledge.

Hinton, A. (2014) Understanding Context. O'Reilly.

ISO (2018) ISO/IEC/IEEE 29148:2018 – Requirements engineering. https://www.iso.org/standard/72089.html.

Jones, P. and Kijima, K. (2018) Systemic Design. Springer.

Morville, P. (2001) The Speed of Information Architecture. https://semanticstudios.com/the_speed_of_information_architecture/.

Polanyi, M. (1966) The tacit dimension. University of Chicago Press.

Resmini, A. and Lindenfalk, B. (2021) Mapping Experience Ecosystems as Emergent Actor-Created Spaces. In Hameurlain, A. et al. (eds) Transactions on Large-Scale Data- and Knowledge-Centered Systems XLVII. LNICS 12630. Springer. doi: 10.1007/978-3-662-62919-2_1.

Resmini, A. and Rosati, L. (2011) Pervasive Information Architecture. Morgan & Kaufmann.

Rovelli, C. (2018) The Order of time. Penguin Books.

Seale, T. (2025) Ontology is having its moment. https://www.linkedin.com/feed/update/urn:li:activity:7400096205990494208/.

Sommerville, I. (2016) Software Engineering. Addison Wesley.

Watcher-Boettcher, S. (2012) Content Everywhere. Rosenfeld Media.

Footnotes

[1] The increased emphasis on the value of relationships over that of individual connected 'things' can be seen as an epiphenomenon of a much broader (and more radical) rethinking of how we perceive reality. This shift likely originates from quantum studies: "the world is made of networks of events, not of things," as the physicist Carlo Rovelli emblematically writes in The Order of Time (originally published in 2017 and translated into English in 2018).

[2] Personal communication, email message to the author, 18 December 2025.

[3] See, for example, Wacher-Boettcher, S. (2012) or Atherton, M. and Hane, C. (2019).

Cite as

Badaloni, F. (2025) Function-Driven Design Method – A Relational Framework for Complex Systems and AI-ready Information Architecture. Journal of Information Architecture. Vol. 08. Iss. 01. Pp. 7–82. http://journalofia.org/volume8/issue1/02-badaloni/. doi: 10.55135/1015060901/261.012/2.050.

Federico Badaloni

GEDI

Federico Badaloni is the current Head of User Experience Design and Graphic Design at the Italian editorial group GEDI. He founded the Executive Master in Information Architecture and User Experience Design degree at IULM University in Milan, and he is a past president of Architecta, the Italian Association for Information Architecture.
Federico holds an ethnology degree and has a background in journalism: after studying piano, violin and clarinet, he completed his composition studies in Rome and has since written several soundtracks for Italian movies and documentaries