The AtLarge Vision on the Design of Distributed Systems and Ecosystems

Alexandru Iosup
Department of Computer Science,
Faculty of Sciences, VU Amsterdam,
The Netherlands
A.Iosup@vu.nl

Laurens Versluis
Department of Computer Science,
Faculty of Sciences, VU Amsterdam,
The Netherlands
L.F.D.Versluis@vu.nl

The AtLarge Team*
VU Amsterdam and
Delft University of Technology,
The Netherlands
http://atlarge.science

***Abstract*—High-quality designs of distributed systems and services are essential for our digital economy and society. Threatening to slow down the stream of working designs, we identify the mounting pressure of scale and complexity of (eco-)systems, of ill-defined and wicked problems, and of unclear processes, methods, and tools. We envision design itself as a core research topic in distributed systems, to understand and improve the science and practice of distributed (eco-)system design. Toward this vision, we propose the AtLarge design framework, accompanied by a set of 8 core design principles. We also propose 10 key challenges, which we hope the community can address in the following 5 years. In our experience so far, the proposed framework and principles are practical, and lead to pragmatic and innovative designs for large-scale distributed systems.**

***Index Terms*—Design; distributed systems; distributed ecosystems; massivizing computer systems; system design; vision.**

I. Introduction

Our knowledge-based society expects a continuous stream of designs—of computer-based services and of the distributed systems on which they run. We use daily many distributed ecosystems [1] whose designs appeared only recently, e.g., of GAFAM and BAT [2], and expect new designs that will lead to considerable economic growth and productivity [3]–[5]. As Figure 1 indicates, design is a common keyword in top scientific and industry venues, including ICDCS. Yet, as we show in this work, we should not take design for granted, and we should not consider that the current approaches will continue to deliver good results. Design problems keep getting more difficult to formulate, and their solutions more difficult to find and reason about. Existing design processes, from merely relying on intuition to classic [6]–[8] to emerging [9], [10], have significant shortcomings for designing distributed ecosystems [1], [11]. Instead, to address this grand challenge of the distributed systems community, we propose a vision toward establishing new theoretical and practical means to produce pragmatic and innovative designs.

> **Definition**: "Design is the intentional solution of a problem, by the creation of plans for a new sort of thing, where the plans would not be immediately seen, by a reasonable person, as an inadequate solution." [12, Loc.345]. *Pragmatic design* is further implemented and runs in production-like settings. *Innovative design* "consists in novel solutions" [13, Loc.2353].

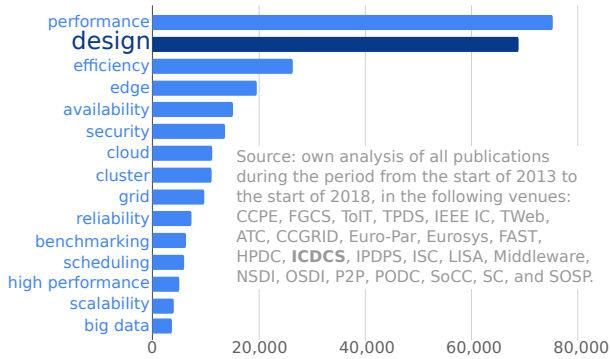

Fig. 1. Presence of selected keywords in top systems venues.

We are interested in a particular kind of design, for *massivizing computer systems (MCS)* [1], that is, for production-ready distributed systems and ecosystems. As in our previous work, we see distributed ecosystems as composites of interconnected (distributed) systems and, recursively, ecosystems. Ecosystems fulfill functional requirements (*FR*s), such as responding to service-queries, or batch-processing big data and computation, and non-functional requirements (*NFR*s), such as predictable high performance and availability. They do so subject to Service Level Agreements (*SLA*s), and in doing so they experience *dynamics*, such as provisioning and releasing resources from an external cloud, and give rise to various *phenomena* that are difficult to foresee at design time, such as performance variability.

> **Vision**: We envision a world of distributed ecosystems, based on pragmatic and innovative MCS designs, created by diverse designers using design philosophy, processes, patterns, and tools, together with scientists, engineers, and the society itself.

We see design as a major challenge for the field of MCS, and raise about it two key questions. *How to find good designs and even good problems?* The ever-increasing complexity of the field—contrast the relatively simple design of the earlier distributed system BitTorrent with the current ecosystems at Google, which can require the orchestration of hundreds of services and systems to produce meaningful results [14], [15]—makes it unlikely that good designs can be achieved from mere

2575-8411/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDCS.2019.00175

sparks of intuition of lonely designers, without good process and collaboration. Not only solving, but also finding problems is increasingly more difficult, and, for ecosystems, finding who should solve them; in contrast, in the 1960s, the core systems problems were well-known, and a small architectural team could direct work on the entire IBM system 360 family [8].

How to design the processes and create the bodies of knowledge that increase the likelihood of good MCS *design?* It is challenging to select the design elements elements that could lead to a high likelihood of good MCS designs, from the hundreds of design patterns [10], [16], [17] and practical steps [18]–[20], and from the many development processes such as rational and agile [9]. Students and even practitioners have rarely studied these systematically, which compounds the problem. But even if the designer would have the experience and knowledge to select, these design elements make many unreasonable assumptions about how designers actually work [11, Ch.3], disregard modern design theory [21] [12, Ch.1-2], and focus not on MCS but on engineering software services [10], software [16], [22], and hardware [8], [23]. In particular, they are not focusing on the daunting systems challenges of new phenomena, emergent behavior, and evolution of technology.

Our vision aims to place design as a core research topic in distributed systems and ecosystems. We do not merely aim to provide a set of *design patterns*, which is a staple of software [24] and of service [10] design but not necessarily the key to design success in distributed systems or even in architecture [25, Loc.572][1]. We also want to steer away from heavyweight design processes, which stifle good design [11, p.233] [25]. We aim to provide a framework for design, from understanding how to think about design in this field to finding and solving MCS design-problems, from design of distributed ecosystems to design supporting experiments of and publications about them, with a five-fold contribution:

1) We are the first to explicitly posit that design is a key area of research in distributed systems, and especially in MCS (in Section II). As support, we offer qualitative and quantitative evidence.
2) We propose the ATLARGE framework for design (Section III). The framework starts from the central premise that design has a fundamentally different nature from science and engineering, which has not been formulated for the distributed systems field. The framework includes novel elements focusing on MCS, about design thinking, problem-finding, problem-solving, and reporting of designs.
3) We propose 8 core principles of MCS design (Section IV). The core principles address four main categories, around the central premise, and systems, peopleware [27], and methodological aspects.
4) We identify 10 current challenges raised by MCS design (in Section V). The challenges are grouped into the same four main categories as the core principles—central premise, systems, peopleware [27], and methodological—and give a broad scope of what the field could address in the next 5 years. Although, in doing so, the community and our own work will supersede the framework elements presented here, we envision the general structure of the framework will be long-lasting.
5) We show evidence, trough real-world experiments, of how the ATLARGE design framework can be pragmatic yet lead to innovative designs (in Section VI), and compare the framework with a multi-disciplinary body of related work (Section VII).

[1]The approach based on design patterns in architecture [26], which has inspired generations of software engineers [24], was quickly dismissed by the architecture community, including by its author, as too limiting.

II. WHY FOCUS ON MCS DESIGN?

We argue in this section for the timely and important need to focus on MCS design. Not only is (good) design needed (Section II-A), but we identify an *increasing* need for good design (Section II-B) and designers (Section II-C). We also analyze what good design needs to address, that is, complex challenges from system design (Section II-D) and from MCS design (Section II-E).

A. Without (Good) Design, We Have Design Debt

Similarly to how Brooks dismissed the idea that organizations can cope with increasing technical debt just by adding more person-months, in this work we want to dismiss the idea that organizations can cope with increasing system complexity (to parallel Brooks, *design debt*) just by hoping good design will simply emerge.

The consequences of not having good designs are well-known, but difficult to quantify. Lackluster design costs money, causes systems to under-perform and sometimes to fail, and delays the arrival of needed systems in the market. Organizations prioritizing working systems over good design effectively defer the moment when they will have to actually solve the problem. In many cases, careful monitoring and capable engineering teams (e.g., sysadmins or site reliability engineers) can help resolve the problems, and in particular avoid unscheduled downtime[2], poor performance, and the resulting bad reputation. However, monitoring only reveals what is measurable and measured [28], leaving organizations exposed to wicked problems (defined in Section II-D) and complex ecosystems (Section II-E).

If lackluster design is costly, bad design can be catastrophic. Design by committee [29] is known to cause entire projects to fail [11, Ch.4], yet many organizations still rely on design by committee done by a central team for technology architecture. A particularly bad case of design by committee is when the entire community ignores the needs of the market and society; fiery arguments in this sense appeared in the databases and grid computing communities, around the start of the 2010s.

[2]Despite recently publishing books on best-practices for distributed systems design [10] and on site reliability engineering [14], [15], since the books were published both the Microsoft and the Google clouds have suffered unscheduled downtime and its related bad publicity.

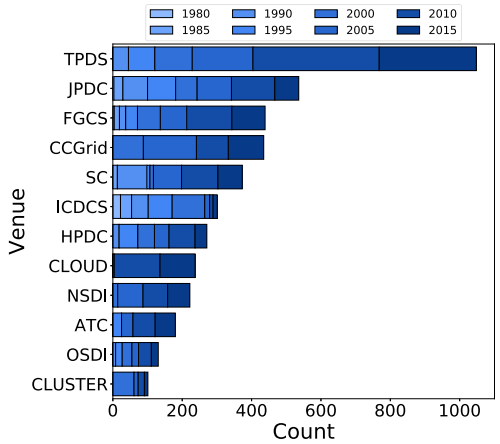

Fig. 2. Count of design articles in selected high-quality computer systems venues, since 1980, in 5-year blocks.

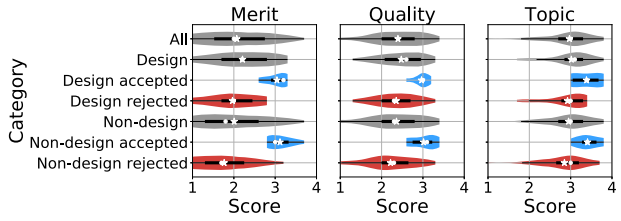

Fig. 3. Violin plots of the scores received by articles in a top quality conference in distributed systems, held in the past 5 years. Articles are grouped in various categories for the purpose of this analysis. Each article is scored for a number of aspects: (**left**) overall merit assessed for each article, (**middle**) quality of contribution, (**right**) match with conference topic. All scores are integers between 1 (lowest) and 4 (highest). (Stars depict averages. White dots depict medians. The thick bar denotes the IQR range. Whiskers show 1.5 × the range, clipped to actual min and max.)

B. The Increasing Need for Good Design

Design articles are increasingly present in major distributed systems venues (Figure 2). Complementing the findings related to Figure 1, we ask *Is the presence of design articles in top distributed systems venues increasing?*

We have extracted all design articles appearing in such venues over a period of nearly four decades (from 1980 to 2018), and counted them per venue and per 5-year block. Figure 2 depicts the count of design articles in selected systems venues, over contiguous 5-year periods starting with 1980. Some of the venues have started earlier, so for them only censured data is available. The last period depicted in the figure, starting in 2015, is incomplete. Many of the venues, including ICDCS, have experienced an increasing accumulation of design articles, with a marked increase in design articles accepted for publication since 2000.

C. The Increasing Need for Good Designers

We also anticipate an increasing need for good designers. We identify two main possible sources for good designers: professionals in the field and students about to become such professionals. We analyze here their capabilities, and conclude there is much room for improvement.

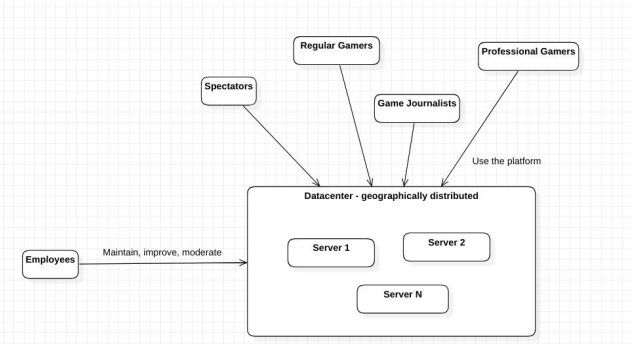

Fig. 4. Typical student design, produced early in a graduate distributed systems course at a top university in computer science. The text is difficult to read, as designed by the student.

Some professionals produce good designs, but still many do not (Figure 3). We analyze for a top conference in large-scale distributed systems all the review-results in one year[3]. For this conference, for each article we have collected whether it is a design article, the final status as accepted for publication in the conference or rejected, and, across the (3+) reviewers, the final scores for (i) the overall quality of the work (the *merit*), (ii) the quality of the approach (*quality*), and (iii) the fit with the topic of the conference (*topic*). Figure 3 depicts the final scores, using distributional (violin) plots. For merit, we find that (1) design articles have a slightly better distributional shape over non-design articles, with higher (better) median, mean, and IQR, and more of the distribution around an overall score of 2 or higher. Across merit and quality, we also find that: (2) a significant percentage of the design articles are not of high quality or high merit (scores significantly below 3). Finding (2) is surprising, because top-tier venues imply self-selection against submitting what the authors themselves consider insufficiently good work; few should merely try out submitting an article to, e.g., ICDCS. This indicates that many professionals still have trouble in both producing and self-assessing their designs.

Graduate students also need training in design thinking and design skills (Figure 4). We analyze here the results obtained from a class of nearly 100 students enrolled in a graduate-level Distributed Systems course[4]; the course seems popular, as the typical class size is around 15 students. We teach in this course not only typical systems concepts from the field, but also concepts and a process for (MCS) design based on the ATLARGE design framework (see Section III). Throughout seven design sessions, students in groups of up to six are tasked to create several designs addressing given problems. Figure 4 depicts an early design, attempting to

[3]We anonymize the venue, but consider it relevant because its held year is after 2014, the venue is a conference, and its ranking is A in CORE18 and green in MSAR14. For comparison, ICDCS has these rankings too.

[4]We anonymize the university, but consider the course relevant because it is large, it took place after 2014, and the university is ranked in the top-150 (in computer science) in both the THE and the QS 2018 World University Rankings (out of nearly 1,000 universities), and in Webometrics of July 18 (out of over 28,000).

satisfice [30, p.27] the problem of scalable ecosystems for massivizing an online game [31]. The figure represents, to a degree, the common submitted design (across all groups) in the same session—what students know after a Bachelors and some graduate courses, but before learning specifically about design. The figure raises many questions about the quality and even the meaning of the proposed design. Even though it is a simplified and high-level design, it still lacks a believable description for solving (even part of) the problem. For instance, an important missing detail are the interconnections, in the geo-distributed datacenter and between stakeholders. This design also lacks any layering, system packaging, or description of the (sub)components. The visual depiction designed by the students is also lacking.

D. New Thoughts on Traditional System Design

System design has gone through successive waves of (shifting) traditional challenges. The 1950s and 1960s system designers were operating in a world where the core problems seemed structured, and the core design approach could be entirely rational, aiming to *optimize* the result [11, Part I]. *Well-structured problems* have several important characteristics [32]: (1) a criterion to automatically evaluate the result, (2) an unambiguous representation for the goal, and start and intermediate states of the problem, and legal transitions between them, (3) a clear representation of all domain knowledge, (4) if interfacing with the natural world, the interaction system-nature can be captured accurately, (5) the problem itself is tractable. By the 1970s, it has become apparent that core problems could further be *ill-structured* [32], that is, not have one or several of the characteristics of well-structured problems, or, worse, *wicked problems* [33], that is, without clear and final formulation, with no universally accepted criteria for success and clearly defined states due to involvement of various stakeholders with competing interests and views, and of various types of hardware and software with various degree of autonomy and limited ability to sense their surroundings. (Now, at the end of the 2010s, the systems community seems to have partially lost this knowledge.[5])

To address ill-defined and wicked problems, the design community has shifted to *satisficing* instead of optimizing designs, and to a process of *co-evolving problem-designs* [11, Loc.935]. A cycle of continuous reaction and adaptation triggers the co-evolution: clients change workloads and SLAs, or laws and standards change; in response, system designers evolve, adapt, and decommission parts of the ecosystem; this triggers another round in the cycle. Co-evolving problem-desings are typical in systems design [1], [11] and pose very significant challenges, in particular because the end-goal is unknowable. For example, Google's datacenter networking evolved significantly over a decade [34], as did Google's Spanner for over 5 years [35].

[5]It is symptomatic for the state of systems design that reviewers in top systems venues do not recognize terms such as "ill-defined" and "wicked", and instead find them too colloquial even after careful explanation and referencing. (We are grateful to ICDCS reviewers for being positively different.)

E. New Challenges in MCS Design

We identify three major trends and related challenges in distributed systems and ecosystems:

(C1) New ecosystem life-cycles: Whereas many past systems were developed and hosted in-house, since 2007 organizations have increasingly shifted operations to (public) cloud computing [3], [5], and thus to distributed ecosystems. Consequently, systems and workloads have become much more fragmented than in the past, requiring new approaches for (automatic) decomposition and orchestration. We identify a strong drive to flexibility and composability at scale; to quote Darwin, "endless forms most beautiful". This raises many new challenges, e.g., the fundamental challenges of MCS [1, S2.2] are about the lack of: (1) operational laws and theories for ecosystems, (2) comprehensive means to maintain existing ecosystems, (3) means to explore credible future ecosystem designs, (4) qualified personnel, (5) adequate inter-disciplinary tools to assess and control the (unwanted) impact of ecosystems on society.

(C2) New ecosystem needs and phenomena: New design aspects appear when designing entire ecosystems or systems operating in ecosystems. In MCS systems have many new NFRs, including various forms of elasticity [36], privacy, interoperability, and operational risk associated with them. Ecosystems are *super-distributed* [1]: they are recursively distributed, with their constituents often being distributed (eco)systems; yet, FRs and NFRs in distributed systems are not known to be directly composable across ecosystems. Various *dynamic phenomena* appear in distributed ecosystems, seemingly unique situations that do not fit the patterns expected from current theory and practice; for example, *vicissitude* [37] is a class of phenomena where several known bottlenecks appear seemingly at random in various parts of the system, *performance variability* is common in clouds [38], datacenter networks [39], and big data operations [40], and ecosystem owners spar with each other (e.g., in Jan 2019, Apple denied Facebook and Google access to its APIs, Unity changed their Terms-of-Service and thus locked out small developers like SpatialOS). Some of these phenomena are due to *emergent ecosystem behavior* and *uncontrolled evolution of technology*.

(C3) New ecosystems, old parts: The evolution of distributed systems technology has generated many useful parts that are commonly used in today's ecosystems, from simple mechanisms (e.g., caching, scheduling), to protocols (e.g., for multi-site data transfer) and policies (e.g., for autoscaling), to relatively simple systems (e.g., BitTorrent for file-sharing), to commonly used architectures (e.g., for web applications, for big data processing). A large amount of legacy applications, using various generations of technology, still operate. Yet, this legacy technology and applications were not designed for the new ecosystems, for example, they are not cloud-native, or they do not match current development and operational processes. Fully replacing them could be prohibitively expensive in the short-term, which means MCS designers must innovate to keep them operational, efficiently.

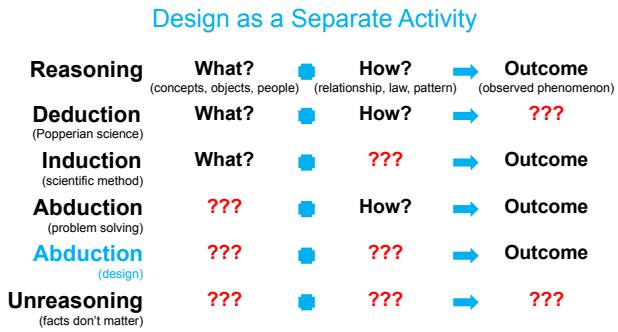

Fig. 5. Dorst considers design an unique intellectual activity, based on abduction [21, p.13]. In particular, design is not science or engineering.

III. The AtLarge Design Framework for MCS

In this section, we summarize the current theories about design as an activity, then focus on the AtLarge design framework. We give its central premise, explain the focus and main concerns, and focus on its key methods for design-space exploration, problem-finding, problem-solving, and reporting to the community and society. Overall, the key contribution of this framework is that it combines current theories about design thinking (Section III-A) with MCS-focused design processes (Sections III-B–III-F).

A. Designerly Ways of Thinking

Design, from engineering component to independence[6]: Ever since the introduction of the concept of "designerly ways of thinking", in the 1990s [25, p.68, concept by Cross], and possibly also earlier, the modern design community has held as a theoretical principle that design is based on specific, idiosyncratic ways of thinking. In 2017, Dorst described a theoretical model for reasoning [21, p.13] that includes design thinking, in which the reasoning universe consists of specific concepts (e.g., real people, software objects), which represent the "What?" of the problem to solve; of relationships between the concepts (e.g., laws of nature, principles of hardware operation, software patterns), which represent the "How?"; and of an outcome that combines the concepts and the relationships (e.g., into a real-world system, into an observable phenomenon).

Figure 5 depicts the Dorst reasoning model. In this model, *deduction* proceeds from given concepts and relationships, and reasons toward an outcome that can be observed (and, thus, testing the deduction); for example, given a Turing machine and a deterministic algorithm designed for it (and its input), we can deduce its outcome. *Induction* follows another classical model from science. Abduction for problem solving (*normal abduction* in Dorst's model) matches well the *software engineering* experience—given the architecture of a software system, determine the best software-design patterns, and the other software engineering concepts and objects, to realize the system that would act as predicted at design-time. *Unreasoning*, which we add to the Dorst model, simply states an extreme of reasoning where any concept, relationship, and outcome can be put together, for example, by an organization for which facts do not matter (one of "alternative facts").

Design abduction: In contrast to the other reasoning approaches in Figure 5, *design abduction* begins with a desirable outcome, and the problem becomes one of finding the concepts and their relationships that lead to the outcome. Of course, an intractable or even infinite number of possible concepts and relationships can exist to consider, which is what makes the design problem rarely amenable to normal abduction (and normal engineering). This does not mean that design abduction must be purely creative, without process.

[6]Computer and software engineering have traversed a similar process until emancipation in the mid-1960s [41, Part III], when detaching from mathematics. Interestingly, mathematics had to follow a similar process, to detach from philosophy; an important part of this process Hilbert's program [42].

Who?	Stakeholders	designers, scientists, engineers, students, society
What?	Central Paradigm	design, different from science and engineering
	Focus	ecosystems, systems within structure, organization, dynamics
	Concerns	functional and non-functional properties; phenomena, evolution
How?	Design Thinking	abductive thinking, processes, co-evolving problem-solution
	Exploration	design space, process to explore
	Problem-finding	structured, ill-defined, wicked
	Problem-solving	pragmatic, innovative, ethical
	Reporting	articles, software, data

TABLE I
An overview of the AtLarge design framework.

B. Overview

We give an overview of the AtLarge design framework and summarize its key properties in Table I: Who? What? How? are the questions addressed in this section.

Who? Stakeholders: The primary stakeholder of MCS design is the society; this is because designs in this field can have an unusually large impact, for a direct product of computing. The AtLarge design framework considers explicitly that designers fulfill a separate role from scientists and engineers, and, consequently, that students require explicit training in design.

What? The Central Premise: design is unique among intellectual activities. Like Cross [43], Dorst [21], and Parsons [12], the AtLarge framework considers design an unique intellectual activity, essentially different from science and engineering. This does not mean that scientists and engineers cannot design—theory and practice indicate all people can and do design naturally [12, Loc.275, theory by Victor Papanek]—, but doing so proficiently and efficiently still requires professional expertise, much like engineering and

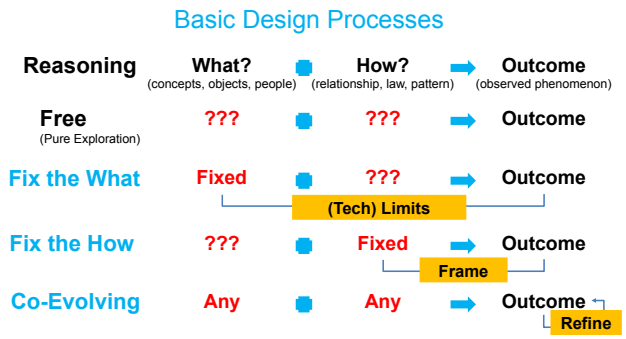

Fig. 6. Basic design processes address how to explore the design space. They range from free to fixed to co-evolving exploration.

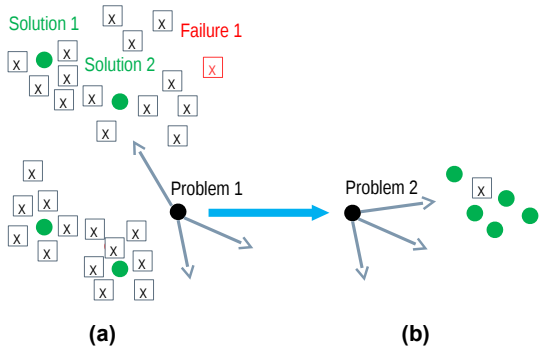

Fig. 7. Design exploration for MCS, through an example. Dark circles are problems. Light (green) circles are successful designs. Boxes marked with an "X" are design attempts that result in failure.

science. Design thinking also addresses emergent system behavior through flexible thought, guided typically by intuition.

What? The Main Focus and Concerns: support for MCS **design.** This requires focusing on both the traditional challenges raised by system designs (see Section II-D) and the new challenges raised by MCS (see Section II-E). Two traditional problems of design are to identify the design space and to explore it efficiently; how to do so for MCS designs is an open challenge. Among the MCS-specific aspects, the ATLARGE design framework considers explicitly, for every problem: the architecture of ecosystems and of systems operating in ecosystems; the structure, organization, and dynamics of ecosystems; functional and non-functional properties and their expression as implicit (that is, designer-given) or explicit (that is, client-given) SLAs; and known aspects of ecosystem phenomena, emergence, evolution.

How? Designerly Thinking: Derived from its central premise, the ATLARGE design framework considers designerly thinking as an essential ability of its practitioners. Among its core elements, this ability includes understanding, conducting, and managing design as co-evolving problem-solutions. In particular, the co-evolving problem-solutions promise to address the emergent nature of systems, by iteratively addressing (emerging) problems. Additional reasoning and practical skills related to science and engineering are also welcome.

How? Key Processes: Although in practice design is still largely an unstructured process, and attempts to impose a rigid structure cause negative reactions [11] and even opposition in software engineering practice[7], the ATLARGE design framework holds that there still is room for (flexible) process for design. Key to good design, the framework proposes not rigid steps, but a small number of flexible methods and processes for: design space exploration (in Section III-C), problem-finding (in Section III-D), a basic cycle for problem-solving (in Section III-E), and for making the results available beyond the design team (in Section III-F).

[7]The agile manifesto, https://agilemanifesto.org/

C. From Free to Co-Evolving Design Exploration

A general, flexible approach to design space exploration for MCS**:** Figure 6 depicts several processes for design exploration. Following the Dorst design framework, the design abduction could be conducted freely, as pure exploration: the designer considers concepts and relationships at will, guided by own intuition and shared community expertise. Although this approach can result in radically new designs, its likelihood of success is limited by the scale of the design space. In contrast, the ATLARGE design process considers three other, more structured approaches for design space exploration. All three consider that there is a process for finding good problems, for example, the process described in Section III-D. The *Fix the What* and *Fix the How* processes explore the same trade-off: they aim to improve the likelihood of obtaining satisficing designs by diminishing the likelihood that the design will be radically innovative. They both do this by limiting the options available to explore. The former does this by fixing the concepts at play and in particular the technology the designer can use; the latter, by fixing the kinds of relationships available to the user ("(re-)framing" in traditional design [21, p.14]).

The third process, *co-evolving*, focuses on iterating designs by changing the problem itself, and further allows using any of the other exploration processes for solving the problem in the current iteration. The staple of this process is the co-evolving problem-solution, with which it can explore a potentially unlimited design space while having a satisficing solution available at each iteration (after the first iteration).

How does co-evolving design space exploration work, in practice? Figure 7 depicts an abstract but realistic example of co-evolving design. The Design Team (DT) is trying to create a pragmatic, innovative design. DT starts with a problem (Problem 1 in Figure 7 (a)). DT creates a design for it, which satisfices or even optimizes the problem (Solution 1). It is not too sophisticated, so DT agrees they could do better. DT tries to do better, and fail (Failure 1). DT learns from it, and produce a new design (Solution 2). Iterating through their design cycle, DT keeps traversing the design space, exploring several dimensions concurrently, and find after much struggle (and failure) a couple more solutions. However, at this point DT concludes it is too difficult and/or costly to keep exploring.

DT has learned enough in the process of design, and possibly with help from their community and clients, and area ready to evolve the problem (Problem 2 in Figure 7 (b)); for example, DT could focus the design on a new ecosystem, replacing the old ecosystem that proved to be too limited for DT to solve the problem. (This does not mean the old ecosystem is not good for other design teams or for other design problems.) It turns out that, for this new problem, DT can find many new solutions relatively easily. The process is successful, and promises more success for the future.

D. Problem-Finding Process for MCS Design

Approach: It would be presumptuous to claim there exists a process for finding all the problems MCS designers can solve. Instead, inspired by how conferences in the field use Calls for Papers to steer the authors, the AtLarge design framework aims to focus the designer by proposing a set of *problem archetypes* (topics). The community could help expand and refine this set in the future. This approach seems highly successful in focusing designers—Figure 3 (right) indicates the designs submitted for evaluation match closely the topics proposed by the conference's community, as proposed by the Program Chairs. Although none of the concepts used in the framework is new, synthesizing these aspects into a catalog, as we do in this work, is novel for the field and seems valuable (see Section VI).

What kinds of problems? Derived from Section II-E, the AtLarge design framework proposes to focus on: (P1) problems in ecosystem life-cycle, including for new and emerging processes and services, and for new and emerging ecosystems; (P2) problems related to new and emerging needs of ecosystem-clients and -operators, addressing newly discovered, emerging, and recurring phenomena, and harnessing new technology (a special kind of phenomenon); (P3) problems related to leveraging and maintaining legacy components. Besides problems that lead to creating new technology, (P4) inspired by natural sciences, where understanding the morphology of natural ecosystems is valued, problems related to understanding how new and emerging technology actually works in practice or when placed in ecosystems, and what new phenomena appear related to ecosystem-operation; (P5) inspired by mathematics, where creating new abstractions can be important regardless of application, problems related to previously unexplored parts of the design space.

How to identify meaningful problems? Also here, the AtLarge design process tries to select from known approaches to identify problems. For addressing problems of types (P1)–(P3), the designer could try to collect and adapt problems from various sources: (S1) (peer-reviewed) qualitative and quantitative studies conducted on ecosystems and on systems within them; (S2) discussion with experts, own analysis of best-practices including reading of technical reports, tech blogs, and best-practice books; (S3) own thought and lab experiments concerning the key technology trends, known technical and other limitations, etc.

For P4, the designer could follow a process matching (empirical) science, but focusing on systems, leveraging the scientific process as finder of phenomena to be harnessed. This could include understanding how systems work through collection and analysis of data archives, where the data represents workloads (e.g., structure of jobs, job life-cycle events such as arrivals, migrations, and cancellations) and operations (e.g., utilization of specific components, (un)availability events). Here, an important set of problems relate to collecting meaningful data: the construction of the observation or measurement instrument, the design of a meaningful data-collection protocol, etc. Currently, these problems seem largely ignored in our field, leading to a dearth of meaningful data for experiments and, possibly, for discovering real problems.

For P5, derived from the notions of views in software engineering [44] and of morphological analysis in sciences [45], the designer could identify unoccupied niches and formulate the problem of exploring them, driven by curiosity.

E. Problem-Solving Process for MCS Design

Approach: Similar to problem-finding, problem-solving is too diverse to capture in any single process; moreover, stage-based processes can raise resistance from practitioners as too constraining [25]. The AtLarge design framework aims to balance the pragmatic need to have a process with clear stages, which allows teams to synchronize about and during the process of design, and the need for innovation that is based on the flexibility to not stifle creativity. To this end, the framework includes an iterative process focusing on creative tasks, which in particular allows its practitioners to skip any step at each iteration. Unlike typical processes in the field, which focus either on hardware design [8], [23] or on software design [16], [22], or on higher-level processes on keeping the team agile [9], the AtLarge problem-solving process focuses first on system-level concerns. Pragmatically, this means it considers first the concepts, components, and challenges specific to MCS.

To manage the complexity of the problem of designing distributed systems and ecosystems, the AtLarge problem-solving process includes two core elements: (1) a *Basic Design Cycle (BDC)*, which is a general process for solving problems, and (2) an *Overall Process* that combines several BDCs into a structure for decomposing and solving MCS design-problems. We only sketch our problem-solving process here, and leave the full detail for future work.

The BDC is the core loop: The BDC process aims to solve any generic design problem through a structured process consisting of the following elements: (1) Formulate requirements, (2) Understand alternatives, (3) Bootstrap the creative process, (4) High-level and low-level design, (5) Implementation of mathematical analysis code, of simulators, of prototypes, etc. (6) Conceptual analysis of the design, (7) Experimental analysis of the design, (8) Result summarizing and dissemination. This approach is by design: it matches many classic design processes, and is recognizable to designers

and engineers in the distributed systems field, yet each element includes key innovations.

The Overall Process (OP) is executed iteratively. It operates as an BDC and, hierarchically, its more complex *design stages* can also operate as BDCs. This design of the OP allows the designers to partition into manageable parts the inherently complex process of solving the problems typical of MCS design, e.g., formulating requirements, creating believable designs[8]. The hierarchical nature of the OP further facilitates learning the process by practitioners: once a practitioner has learned the BDC, they can apply it several times in the OP.

The OP has one more important feature: in each iteration, each of its stages can be skipped as needed. By not forcing the designer to traverse unnecessary elements, the OP allows each iteration to be tailored to the remaining parts of the problem to be solved, and to the remaining time and other resources. We conjecture this can lead to designerly thinking (see Section III-A).

The OP elements: Figure 8 depicts the OP. Given a design problem, its BDC spans elements 1–8, with various groupings allowing for finer-grained iteration. In the overall BDC process, elements (5) and (7), which can include various types of prototype implementation and of experimentation, respectively, can be complex. When this complexity occurs, the designers need to expand them each into one BDC. Similarly, Element (8), on reporting, engineering, and public dissemination, can further expand into separate BDC processes for publishing articles, free open-access software (FOSS), and FAIR [46] or free open-access data (FOAD); we explain this element in Section III-F.

Stopping criteria: As any *iterative* process, BDC stops when meeting a predefined set of: (1) finding a single answer that *satisfices* [30, p.27], that is, gives solutions that are "good enough", or, where possible, *optimizes*; (2) finding a few answers, forming a portfolio to allow a human reviewer (e.g., a client) to quickly select one; (3) finding many answers, forming a *systematic design*, that allows an expert reviewer or system to select one; (4) finding all answers, resulting in *design space exhaustion* and allowing experts across the community to discuss or select results; (5) running out of time or other resources (e.g., funding).

BDC can, but does not guarantee success: Because it admits the stopping criterion 5, the BDC does not guarantee a result. In our experience so far, following the OP process has a good probability of success, making pragmatic and innovative designs likely within the time- and resource-budget. We present experimental evidence for this in Section VI.

[8]That the result is *believable* is the core of the epistemological problem of design [12, Loc.972]. A design that does not solve the problem believably is not acceptable as good design; in our experience, this argument is commonly made to reject articles that propose designs. This is even more important for MCS-designs, because such designs are unlikely to be analyzed experimentally to the full extent of their intended application. In other words, many designs will *at best* be shown as believable, through narrow laboratory experiments, so they must be at least believable to add value to the community.

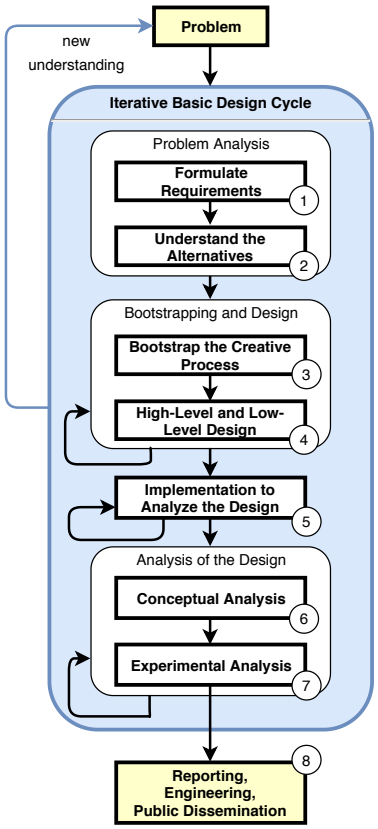

Fig. 8. The ATLARGE design process.

F. Dissemination Processes for MCS Design

The ATLARGE design framework also considers various forms of dissemination typical for MCS, related to reports, software, and data. Each of these means of dissemination is based on some form of design; for example, designing the reports to be published as peer-reviewed articles. Thus, for each, the framework proposes design-based processes; in essence, smaller versions of the framework itself, and in particular of the BDC (see Section III-E).

Good dissemination should increase the likelihood of good designs. Although the dissemination of reports, software, and data can be achieved through much intuition, expertise, and by following best-practices in the respective fields, in practice many of the designs for dissemination are poor. Best-practices include (too?) many tools, e.g., collaborative editing using a tool such as Overleaf; collaborative FOSS development using CI/CD tools such as Travis CI and customized solutions [47]; and sharing code and data on archives such as GitHub and Zenodo, respectively. The quality of dissemination practices has not been studied extensively, but the symptoms are telling, as introduced in Section II-C. Moreover, we have observed [48] cases of design under-specification, that is, specification that does not permit the reproduction of either the designs themselves or of their intended results, even with adequate expertise.

Type (Section)	Principle Index	Key aspects
Highest (§IV-A)	P1	design of design
Systems (§IV-B)	P2	age of distributed ecosystems
	P3	NFRs, phenomena
	P4	RM&S, self-awareness
Peopleware (§IV-C)	P5	education in design
	P6	pragmatic, innovative, ethical
Methodology (§IV-D)	P7	design science, practice, culture
	P8	evolution and emergence

TABLE II
THE KEY PRINCIPLES OF MCS DESIGN.

IV. Design Principles of MCS

We introduce in this section a set of core principles for MCS design. Table II summarizes the principles.

A. Highest Principle

P1: Good design processes foster good system designs.

We have argued for this principle in Section II. The highest design principle holds that MCS design must be designed, not left only to intuition and selective experience.

B. Systems Principles

P2: This is the Age of Distributed Ecosystems.

As stated in Section II-E, the evolution of distributed systems into ecosystems led to important new problems and solutions. This principle argues for an approach to design where the designer is constantly aware of this fact.

P3: Dynamic non-functional properties and phenomena are first-class concerns.

P4: Resource Management and Scheduling, and its interplay with various sources of information to achieve local and global Self-Awareness, are key concerns.

Principles **P3** and **P4** are consequences of MCS problems always including dynamic and emergent elements. Good MCS designs must consider complex SLAs, emergent phenomena, information-rich decision-making, etc.

C. Peopleware Principles

Popular distributed ecosystems service hundreds of millions daily. It is not uncommon for a typical service to call into execution hundreds of hidden systems. This combination of high complexity and responsibility puts pressure on the human resources—the peopleware.

Inspired by the software industry's struggle to manage and develop its human resources, we explicitly set principles about peopleware.

P5: Education practices for MCS must ensure the competence and integrity needed for experimenting, creating, and operating ecosystems.

Because the complexity and responsibility of the job has increased considerably over the past couple of decades, high-quality design education should become a core principle of MCS. With proper training, the community will remain able to produce designs significantly better than early, student-like attempts (see Figure 4), and avoid a culture of hacking that does not work long-term. Education on the ethics of design is also a must, if the community is to avoid even the most basic traps, such as engendering bias and disregarding privacy.

P6: Design communities can foster and curate pragmatic, innovative, and ethical design practices.

The community is already structured to foster and curate designs (see Section II-B). This principle extends this structure to include shared tools and environments for developing and evolving designs: shared datasets and benchmarks, testing infrastructure available to many, common repositories of and documents about operational patterns, online virtual laboratories for global coursework and training, etc. These are elements that greatly facilitate design, and make it pragmatic by linking academia and industry. The community is also best-equipped to understand and explain the ethics of the field, and further to handle ethical risks.

D. Methodological Principles

P7: We understand and create together a science, practice, and culture of MCS design.

So far, design has not been treated as a scientific subject in the field of distributed systems. However, design should become such a subject, because it meets the requirements explained by Denning [49, p.32]: (i) MCS design is a pervasive phenomenon, which we try to understand, use, and control; (ii) both artificial and natural processes are at play (designs lead to real-world artifacts); (iii) we aim to gain meaningful and non-trivial understanding of the phenomenon; (iv) we aim to make our findings reproducible, so that good designs become more likely, consequence of falsifiable theories and models; etc.

P8: We are aware of the history and evolution of MCS designs, key debates, and evolving patterns.

Unlike other exact results in distributed systems, design is prescriptive, and often discursive. This makes it subject to debate and interdisciplinary expertise. To improve design, we need to make use also of the key instruments of empirical research, including exploring the history of the field, surveying the expert view, understanding the key debates and their ongoing resolution (as Tedre does for the whole field of computer science [41]), etc.

Type (Sec.)	Challenge Index	Key aspects	Pr.
Highest	C1	Design of design	P1
Principle	C2	(∼) What is good design?	P1
(§V-A)	C3	(∼) Design space exploration	P1
Systems	C4	Design for ecosystems	P2
(§V-B)	C5	(⋆) Catalog for MCS design	P3–4
Peopleware	C6	(⋆) Education, curriculum	P5
(§V-C)	C7	(⋆) Community engagement	P6
Methodology	C8	(⋆) Documenting designs	P5–8
(§V-D)	C9	(∼) Design in practice	P7–8
	C10	(⋆) Organizational similarity	P7–8

(⋆) Appears only in the extended version [50].
(∼) Summarized here, see extended version [50].

TABLE III
A SHORTLIST OF THE CHALLENGES RAISED BY MCS.

V. TEN CHALLENGES FOR MCS DESIGN

Many challenges must be overcome before the principles in Section IV can give us a solid basis for design. Known challenges begin with making the highest principle, of MCS design being based on a design rather than on intuition, a reality. Challenges appear also related to systems, peopleware, and methodological aspects. We give in the following a non-exhaustive list of ten challenges for MCS design.

A. Challenges Related to the Highest Principle

C1: The design of design [11]. Creating processes to enable and facilitate pragmatic and innovative MCS designs.

The diversity of already existing design processes (see also Section VII) can come as a surprise to the MCS designer, and even to the best of system designers [11, Part I]. Yet, the challenge of designing the MCS design remains open.

First, as we explain in Section VII, much exploration, combination, and innovation is still possible. The framework we propose in this work has been tested only by one research group, albeit large and long-lasting; new designs of (MCS) design could prove vastly superior.

Second, as the following challenges indicate, we have not yet understood the full extent of the problem raised by MCS design. We envision new aspects will become relevant, leading to a co-evolving problem-solution.

C2: (∼) Understand what is good design.

Currently, the community relies largely on human experts to assess and curate designs. (In contrast, in hardware design, design space exploration has been largely automated.) We pose as an open challenge understanding (automatically) what is good design. This is not easy.

First, top venues use criteria such as "degree of innovation" and "quality of the approach", but their discrete formulation may ask reviewers to overfit their assessment to a quantitative estimation. Consequently, as exemplified in Figure 3, many scores cluster around the middle of the given range, leading to difficulties in separating the better designs from their near-equivalents. What alternative approach could be used?

Second, reviewers often also introduce in their assessment other criteria that have never been analyzed thoroughly. For example, simple designs are valued, which seems reasonable because simple designs foster system maintainability; but the evidence simplicity is the right trade-off between the quality of the approach and maintainability, or even a common understanding of what makes a system simple, are lacking. Other criteria, such as balance of the approach or another (semi-)aesthetic aspect (e.g., "elegant design"), have also not been studied. This contrasts with the nature of real-world ecosystems, which are messy by nature, and which combine various designs created by different organizational cultures.

C3: (∼) Simulation-based approaches and experimentation for design space exploration. Calibration and reproducibility are key.

In maze-solving, it is known that finding an exit is much harder when the alternatives are numerous than for a straight path. Yet, in our field, the complexity and the number of alternatives considered and eliminated before the design has emerged, or more broadly the characteristics of the design space, are rarely discussed in our articles or by their reviewers. How to characterize the broad and diverse design spaces available in MCS design?

B. Systems Challenges

C4: Design for MCS, not for individual systems.

We see as the grand challenge of MCS design to understand how the resulting design will fit in an entire ecosystems. Typical questions include: How to enable and how to future-proof the design of systems that need to interoperate, especially dynamically, at runtime? For example, how to enable cross-cloud operation, service delegation and federated composition, and geo-distributed data use? Is this even achievable with high likelihood of success, when ecosystems combine organically designs from different organizations and business units, and thus suffer the consequences of Melvin Conway's empirical law [29] that designs "by committee" are likely to fail?

Current approaches already reveal patterns in the core topics pursued by the community. These include [1]: (i) adaptation and self-awareness in ecosystems, (ii) ecosystem navigation: find and solve common problems of comparison, selection, composition, replacement, adaptation, and operation; (iii) discovering the new world: creating designs responding to new modes of use; (iv) the challenge to support non-functional requirements (see **P3**); (v) the ecosystem-scheduling challenge: design scheduling approaches to be flexible enough to represent MCS needs, diversity, and heterogeneity, and solve both the provisioning and the allocation problems.

Addressing this challenge could also start from understanding the workload and relative importance of individual components in current ecosystems. This could give quantitative evidence that some components are naturally more important than others, and thus focus the community efforts. One of the likely steps in this sense is to observe pragmatically which part of the current ecosystem is taking much engineering time, and re-design that part into "⋆-as-a-Service".

C5: Establish a catalog of components for MCS design.

Such a catalog would consist of design principles, known architectural and operational patterns, etc. Useful catalogs are a known approach for settled fields [51], [52], but how to build a useful catalog for MCS designs?

C. Peopleware Challenges

C6: (⋆) Create a teachable common body of knowledge for MCS designs, focusing on pragmatism, innovation, and ethics. Design effective teaching practices for this curriculum.

C7: (⋆) Create communities and environments for people to engage with the design and operation of ecosystems, to demonstrate and explain operations.

D. Methodological Challenges

C8: (⋆) Design a formalism for documenting designs.

C9: (∼) Understand MCS design in practice. How and when do MCS practitioners design what they design?

We know normal abduction is commonly used in engineering [13, Ch.5] [11, Part I], especially coupled with complex implementation and realization processes [8]. However, the extent and approach of using design abduction in practice are not currently known. The "When?" is also important; for example, design used to be a static process done at the start of projects, but in some dynamic organizations it is now part of the weekly sprints and helps the DevOps teams respond quickly. In consultancy, design may encounter strict time constraints, and also need to address unusual requirements.

C10: (⋆) Organizational similarity in MCS design.

VI. Experiments with the Design Framework

We have used the AtLarge design framework as our main approach to design for the past decade. Effectively, we have designed its various processes, conducted experiments with them, and refined them as we uncovered the many problems a research group faces in creating pragmatic and innovative MCS designs. (Our designs are also ethical, as assessed by our reviewers, institutions, and funding agencies.)

We summarize in Table IV our use of the AtLarge design framework for over a decade, for a broad range of MCS designs: (1) co-evolving understanding, and protocol and system design for Peer-to-Peer systems [50, §6.1]; (2) design for online gaming ecosystems [50, §6.2], as an example of designing in rapidly changing ecosystems operating under strict NFRs; (3) design for datacenter ecosystems [50, §6.3], as an example of evolving understanding of how the field's reference architecture is emerging; (4) design for serverless ecosystems [50, §6.4], as an example of how the AtLarge design process fosters collaboration between diverse teams; (5) the design of the Graphalytics ecosystem [50, §6.5], as an example of DevOps support; (6) design for portfolio scheduling [50, §6.6], as an example of co-evolving a detailed system-level design; (7) the design of experiments in autoscaling [50, §6.7], as an example of designing both real-world and simulation-based experiments.

Section	Experiment	Key aspects
(⋆) §6.1	P2P	Protocol/Sys. design
(⋆) §6.2	MMOG	Ecosystem, NFRs
(⋆) §6.3	DC management	RM&S, ref.archi.
(⋆) §6.4	Serverless, FaaS	Design in new ecosystem
(⋆) §6.5	Graphalytics	Ecosystem, DevOps
(⋆) §6.6	Portfolio scheduling	System design
(⋆) §6.7	Autoscaling	Experiment design

(⋆) Appears only in the extended version [50].

TABLE IV
Experiments with the AtLarge design framework.

Overall, we conclude the AtLarge design framework passes the following criteria for success:

1) It allow us to co-evolve problems and their solutions, even for problems with very successful solutions, or for very challenging problems with no or few solutions;
2) It help us identify "hot" problems, and make scientific discoveries with impact on the community;
3) It enables us to create pragmatic and innovative designs, as assessed by our own team and by the expert reviewers;
4) It keeps our design activity fit to receive competitive funding from academic and industrial funding organizations, and interesting and motivating to attract a diverse group of young researchers;
5) It results in publications accepted by high-quality venues, which we see as proxies of high-quality designs and results, and foster other useful results ancillary to good design practices (e.g., publishing high-quality software and data artifacts).

We also conclude that the design methods from software engineering, including SCRUM and agile methods [9], albeit useful, do not cover the system-level elements needed in the design of distributed systems and ecosystems.

VII. Related Work

In this section, we compare our and related work. We expand greatly this section in our technical report [50], by further comparing this work with design in computer systems, in software engineering, and in general.

Overall novelty: The ATLARGE design framework combines elements of 2010s design thinking with the specifics of MCS design. The former makes it unique among published design frameworks in distributed systems. For example, hardware design is a well-established field of design, but as noted by Brooks it has not adopted the new ways of design thinking [11, Part I]. The latter makes it unique among design frameworks. For example, works of similar scope address the design of mechanical systems [51], [53], but their physical properties makes them radically different from distributed systems.

VIII. CONCLUSION

Responding to the needs of an increasingly digital and knowledge-based society, in this work we explicitly posit that design is a key area of research for distributed systems and ecosystems (MCS), and propose a vision to establish the theory and practice of MCS design.

Ours is the first attempt to understand the problem of MCS design. We give qualitative and quantitative evidence of the extent of the problem, and propose requirements derived from general design processes and from the specific needs of MCS.

We design the ATLARGE design framework around the central premise that design is fundamentally different from science and engineering, requiring its own way of thinking and processes. Responding to requirements, the framework combines emerging theories about design thinking with several MCS-focused design processes, e.g., for co-evolving problem-designs, for problem-finding and -solving, and for disseminating the results. We have a decade-long experience with this framework; in our experience, it can lead to pragmatic and innovative designs in diverse fields: P2P systems, datacenter ecosystems, cloud computing, MMOGS, graph processing, etc.

Our vision also includes a set of core principles and challenges of MCS design, in the four broad categories related to the central premise, systems, peopleware, and method. We have started to address the research agenda formulated in this article. We hope this vision will stimulate a larger community to join us in improving the design *processes* used in distributed systems and ecosystems. Everyone (trained) can design!

ACKNOWLEDGMENTS

We thank NWO Vidi MagnaData and our reviewers.

CO-AUTHORS

* This article is the result of extensive discussions in the AtLarge research team about the nature of design processes in distributed systems and ecosystems. The full list of authors for this article is: Alexandru Iosup, Laurens Versluis, Animesh Trivedi, Erwin van Eyk, Lucian Toader, Vincent van Beek, Giulia Frascaria, Ahmed Musaafir, Sacheendra Talluri.

REFERENCES

[1] Iosup *et al.*, "Massivizing computer systems: A vision to understand, design, and engineer computer ecosystems through and beyond modern distributed systems," in *ICDCS*, 2018.
[2] The Economist, "Taming the titans," Jan 20–26, 2018, p.11–12, https://www.economist.com/printedition/2018-01-20, Jan 2018.
[3] European Commission, "Uptake of cloud in europe," Digital Agenda for Europe report, Sep 2014.
[4] ——, "Big Data and data analytics," EU Parliament, Sep 2016.
[5] Gartner Inc., "Infrastructure and Operations (I&O) Leadership Vision for 2017, section CIO Technology Priorities," Tech.Rep., 2017.
[6] Royce, "Managing the development of large software systems," in *IEEE WESCON*, 1970.
[7] Boehm, "A spiral model of software development and enhancement," *IEEE Computer*, vol. 21, 1988.
[8] Blaauw & Brooks, *Computer Architecture*. Addison-Wesley, 1997.
[9] Ramsin & Paige, "Process-centered review of object oriented software development methodologies," *ACM Comput. Surv.*, 2008.
[10] Burns, *Designing Distributed*. O'Reilly, 2018.
[11] Brooks, *The Design of Design*. Addison-Wesley/Pearson, 2010.
[12] Parsons, *The Philosophy of Design*. Polity, 2015.
[13] Arthur, *The Nature of Technology*. Free Press, 2009.
[14] Beyer *et al.*, *Site Reliability Engineering*. O´Reilly, 2016.
[15] ——, *Site Reliability Workbook*. O´Reilly, 2018.
[16] Abbott and Fisher, *The Art of Scalability*. Addison-Wesley, 2015.
[17] Erl *et al.*, *Cloud Computing Design Patterns*. Prentice Hall, 2015.
[18] Lidwell *et al.*, *Universal Principles of Design*. Rockport, 2010.
[19] Boeijen *et al.*, *Delft Design Guide*. BIS Publishers, 2014.
[20] Martin & Hanington, *Universal Methods of Design*. Rockport, 2012.
[21] Dorst, *Notes on Design*. BIS Publishers, 2017.
[22] Bass *et al.*, *DevOps*. Addison-Wesley, 2015.
[23] Hennessy & Patterson, *Computer Architecture*. Morgan Kauf., 2017.
[24] Gamma *et al.*, *Design Patterns*. Addison-Wesley, 1994.
[25] Lawson, *How Designers Think*. Taylor and Francis, 2004.
[26] Alexander *et al.*, *A Pattern Language*. Oxford U. Press, 1977.
[27] DeMarco & Lister, *Peopleware*. Dorset House, 2012, 1st Ed. 1986.
[28] Bouwers *et al.*, "Getting what you measure," *CACM*, 2012.
[29] Conway, "How do committees invent?" *Datamation*, vol. 14, 1968.
[30] Simon, *The Sciences of the Artificial*. MIT Pess, 1996.
[31] Shen *et al.*, "Massivizing Multi-player Online Games on Clouds," in *CCGrid*, 2013.
[32] Simon, "The structure of ill structured problems," *Artif. Intell.*, 1973.
[33] Rittel & Weber, "Dilemmas in a general theory of planning," *Policy Sciences*, vol. 4, 1973.
[34] Singh *et al.*, "Jupiter rising: a decade of clos topologies and centralized control in google's datacenter network," *CACM*, vol. 59, 2016.
[35] Corbett *et al.*, "Spanner: Google's globally distributed database," *ACM Trans. Comput. Syst.*, vol. 31, 2013.
[36] Herbst *et al.*, "Quantifying cloud performance and dependability: Taxonomy, metric design, and emerging challenges," *TOMPECS*, vol. 3, 2018.
[37] B. Ghit *et al.*, "V for Vicissitude: The Challenge of Scaling Complex Big Data Workflows," in *CCGrid*, 2014.
[38] Iosup *et al.*, "On the Performance Variability of Production Cloud Services," in *CCGrid*, 2011.
[39] Ballani *et al.*, "Towards predictable datacenter networks," in *SIGCOMM*, 2011.
[40] Uta & Obaseki, "A performance study of big data workloads in cloud datacenters with network variability," in *ICPEW*, 2018.
[41] Tedre, *The Science of Computing*. CRC Press, 2015.
[42] Franks, *The Autonomy of Mathematical Knowledge: Hilbert´s Program Revisited*. Cambridge University Press, 2009.
[43] Cross, *Design Thinking*. Berg, 2011.
[44] Rozanski & Woods, *Software Systems Architecture*. Addison-Wesley Professional, 2005.
[45] Zwicky, "Morphological astronomy," *The Observatory*, vol. 68, 1948.
[46] Wilkinson *et al.*, "The FAIR Guiding Principles for scientific data management and stewardship," *Nature SciData*, vol. 3, 2016.
[47] Widder *et al.*, "I'm leaving you, travis: a continuous integration breakup story," in *MSR*, 2018.
[48] Andreadis *et al.*, "A reference architecture for datacenter scheduling: design, validation, and experiments," in *SC*, 2018.
[49] Denning, "The science in computer science," *CACM*, vol. 56, 2013.
[50] Iosup *et al.*, "The atlarge vision on the design of distributed systems and ecosystems," *CoRR*, 2019, published Feb 2019. [Online] Also available: https://atlarge-research.com/pdfs/mcsdesign19tr.pdf.
[51] Pahl *et al.*, *Engineering Design*. Springer-Verlag, 2007.
[52] Bell *et al.*, *Computer Engineering*. Digital Press, 1978.
[53] Ullman, *The Mechanical Design Process*. David Ullman LLC, 2017.