Controlled Randomness

Shape Generation within the Branding System of Politecnico di Milano

Author: Sara Stojanovic
Supervisor: Assoc. Prof. Marco Quaggiotto

Politecnico di Milano
Laurea Magistrale in Communication Design
A.Y. 2024/2025

Controlled Randomness

Shape Generation within the Branding System of Politecnico di Milano

Author: Sara Stojanovic
Student ID: 245293
Supervisor: Assoc. Prof. Marco Quaggiotto

Laurea Magistrale in Communication Design

Politecnico di Milano
A.Y. 2024/2025

Abstract

Con la crescente applicazione dei sistemi generativi, in particolare delle piattaforme di AI sempre più diffuse nel design e adattate al branding e al linguaggio visivo, che permettono livelli crescenti di automazione, restano aperte importanti questioni etiche, così come interrogativi sull'autorialità, unicità, trasparenza e sull'imperfezione umana che conferisce un senso di realtà.

Da qui nasce la motivazione iniziale di questa tesi: applicare il design computazionale al branding del Politecnico come forma di automazione parziale, con l'intento di preservare la scelta umana, la navigazione del designer e l'allineamento con l'identità visiva del brand, che così può espandersi mantenendo la propria narrazione.

Attraverso il lavoro svolto durante il tirocinio presso l'Ufficio Comunicazione dell'Ateneo, senza le cui indicazioni questo progetto non sarebbe stato possibile, è stato sviluppato un programma, un generatore di elementi basato sul sistema circolare di forme già definito nel brand book, utilizzato come pattern ed elemento di supporto per materiali stampati e media digitali.

Il programma è concepito come un sistema di forme radiali e poligonali con parametri controllati, in cui ogni utente può ottenere una forma con il numero di lati desiderato, che ruota, si arrotonda, assume angoli vivi o smussati, generando un ampio spettro di configurazioni circolari. In questo modo, un'istituzione caratterizzata da un design coerente acquisisce elementi secondari più giocosi e la possibilità di soluzioni non ripetitive, pur restando all'interno della narrazione visiva.

Tuttavia, la varietà dei parametri comporta anche un limite: alcune forme, pur essendo circolari e formalmente corrette, non risultano compatibili con il brand istituzionale. Questo limite ha portato al miglioramento del processo di generazione casuale attraverso la valutazione degli oggetti prodotti da parte dell'Ufficio Comunicazione, dell'autrice e del relatore, tramite un sistema di rating da 1 a 5. I rating raccolti con i parametri sono stati usati per l'analisi e per individuare pattern negli output più valutati, riducendo la casualità e favorendo risultati più applicabili al branding, lasciando spazio a sviluppi futuri.

Abstract

With the increasing application of generative systems, especially AI platforms that are gaining widespread use in design and are being adapted to branding and visual language, enabling growing levels of automation, traces of major ethical questions remain, as well as questions of authorship, uniqueness, transparency, and human imperfection that gives a sense of reality.

This leads to the initial motivation of this thesis, which is the application of computational design in the branding of the Politecnico institution, as a form of partial automation, but with the intention of preserving human choice, designer navigation, and alignment with the visual identity of the brand, which gains the possibility of expansion while preserving its narrative.

Through engagement during an internship in the University's Communication Office, without whose indications this project would not have been possible, a program was developed, a generator of objects based on the already existing circular system of shapes defined in the brand book, which are used as patterns and supporting elements of printed materials and digital media.

The program is conceived as a system of radial and polygonal shapes with controlled parameters, where each user can obtain a shape with as many sides as desired, which rotate, become rounded, gain sharp or rounded corners, forming a wide spectrum of circular shapes. In this way, an institution characterized by consistent design gains playful forms of secondary elements and the possibility of design that does not repeat itself, while always fitting within the narrative.

However, the diversity of parameters also carries a drawback: certain shapes, even if circular and formally correct, are not compatible with the institutional brand. This drawback led to the improvement of the random generation process through the evaluation of generated objects by the Communication Office, the author, and the supervisor, using ratings from 1 to 5. The collected ratings with parameters were used for analysis and for identifying patterns among highly rated outputs, reducing complete randomness and enabling more frequent results that are applicable within the branding, while leaving space for further development.

Introduction

Over the past century, design has evolved from the creation of individual artifacts into a practice increasingly concerned with systems, processes, and the management of complexity. Design is shifting from fixed solutions toward systems that produce variations, but this change raises questions of control and responsibility. There are segments of design that readily accept such changes and engage with innovation, with greater access to new technologies, and those that consistently remain a few steps behind, due to safety, inflexibility, and a lower level of trust in new methods. When it comes to institutional branding, it is necessary to find a balance. Institutions that engage with innovation should also brand themselves through the expansion of design novelties, while maintaining consistency and visually communicating trust in both the present and the future of the creative world.

Every strategy of visual representation and identity communication is based on clearly defined methods and steps. For example, a simple poster announcing an event for a company or institution must follow the brand manual's rules for composition, tone, color, typography, contrast, and hierarchy. (Cullen, 2012) These items are not chosen freely; they are part of a system that guides how the identity should appear across various situations.

For an institution to achieve a strong visual response, its visual narrative requires following a set of established rules and ways of application, with clear answers to why certain choices are made, how they are applied, and what their purpose is. (Humaid et al., 2025) For the viewer, when encountering a single piece of communication, this structure is neither obvious nor consciously perceived. However, over time, this soft presence, through repetition, consistency, and visual strength, gains meaning and leaves a trace.

At first, this may sound simple: a set of rules to be followed. Yet it is never like that. Rules may represent the core of an identity, but the real difficulty arises when those same rules leave insufficient space for extension, adjustment, or reinterpretation. Branding doesn't exist in static conditions; it operates in a state of continuous change, new media, new formats, and shifting modes of communication.

The institutional branding of a comprehensive university cannot rely solely on established solutions. It requires a system that is both stable and open, capable of preserving consistency while permitting variation. The relationship between structure and flexibility becomes particularly significant as design shifts from producing singular outcomes to defining adaptable processes.

In this context, computational design is introduced alongside existing manual programs. While computational design does not alter foundational branding principles, it has the potential to expand them.

By translating constraints, relationships, and decisions into systematic frameworks, computational methods enable variation to emerge without disrupting the visual logic of the identity. Thus, computational approaches complement institutional branding by providing additional means for extending and exploring branding systems.

This doesn't mean making things random or losing control. Many modern identity systems already use variation, following clear, simple rules. Designers combine, arrange, change, or repeat elements in specific ways, creating many different results that still look related to the original idea.

Variation is working within set limits. We can add the ability to duplicate objects, rotate them, change their positions, or create a whole range of more diverse shapes. But their identity doesn't change; only how they appear does.

These methods show that you can plan for variety instead of making it up as you go. By setting rules for how things change, designs can stay connected but still adapt. This approach focuses less on the final look and more on how the design is made, which helps create systems that are both organized and flexible.

Computer-based design becomes important when, instead of adding something new, it fits in with the way designers already use rules and changes. Coding makes these steps clear, easy to repeat, and capable of handling more complex projects, so designers can manage complicated work without losing control or style.

In the rebranding of the visual identity of the Politecnico di Milano University, we see a new form of flexibility alongside strength in the institution's rules. Variations are present, and there is room for new ones, especially across different university departments.

System design remains official but is based on principles that can be adapted to different situations. The identity remains clear but adaptable to modern communication needs (Fig. 1). The balance between structure and flexibility lets new designs grow out of the identity, not break away from it. This kind of openness, however, also introduces a series of design challenges.

This research, therefore, engages with the problem of how computational systems can be used to extend institutional branding frameworks, introducing variation while maintaining visual coherence and designer control.

In this context, design decisions are understood as a way of structuring and executing established rules through a generative system. Such a system allows visual outcomes to be produced through predefined parameters and relationships, while leaving selection, evaluation, and responsibility in human hands.

"Creativity does not mean improvisation without method."
— Bruno Munari

Computing here does not remove designers from the process; instead, the thesis focuses on the design of a generative system grounded in existing branding principles and supported by human evaluation. The exploration of adaptive or learning-based approaches is positioned as a direction for future development. The following chapters examine the context, development, and evaluation of this system.

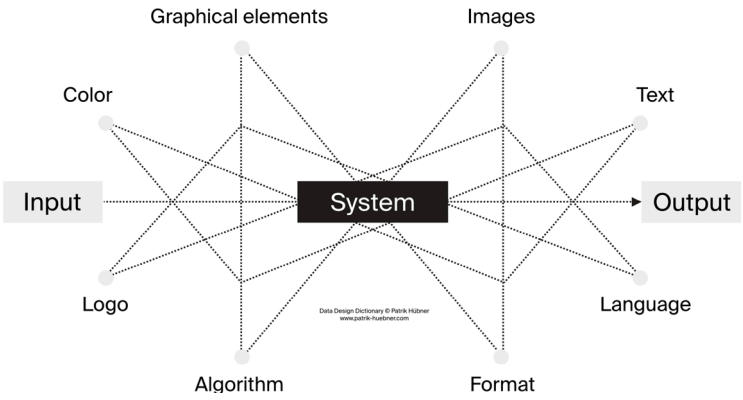


Fig. 1. System diagram illustrating input–system–output relationships. Source: Hübner (n.d).

PART I

Institutional & Practice Context

01

Branding Systems and Institutional Context

Branding, specifically institutional branding, is viewed in this chapter as a dynamic system rather than a fixed visual identity. Brand is not a one-off design; it is a collection of rules, relationships, and strategies that guide how visual elements are expressed over time and in various contexts. The aspiration is to recognize that modern branding is about identity, not just a single visual element, but about creating a cohesive, recognizable, and adaptable system (Wheeler, 2018; Johnson, 2016). There is a need for a unique balance, and the display of authority and stability must fit into the flexibility and changes in communication methods, media, platforms, and organizational needs. Unlike commercial brands, which embrace change and experimentation, institutions maintain a consistent, transparent, and coherent identity that endures over time. Design today must navigate complex social and organizational systems, which means that changes happen thoughtfully rather than abruptly (Manzini, 2015). That is why the institution becomes an ideal setting for researching system-based design approaches, as an interesting link between stability and adaptability. Focusing on how rules, procedures, and repetition lead to brand consistency, we highlight it as a structured system. Contemporary practices are well aware of automation and tool-driven workflows (templates, design systems, and digital platforms), but often without reflecting on their rationality (Johnson, 2016). This chapter uses the visual identity of the Politecnico di Milano as a case study, emphasizing aspects that highlight the potential for computational enhancements. Understanding the branding system now extends beyond manual processes, laying the groundwork for discussing computational design practices.

1.1 Branding as a System, Not a Static Identity

To continue the concept of spreading modern branding and change across different media and technologies, it is necessary to establish a shared understanding of branding within this research. First, we define the difference between brand identity, visual identity, and branding itself. And if these terms are constantly confused in the practice of graphic and communication design, and the same meaning is sometimes assigned to them, they actually refer to other dimensions of the construction of visual communication.

Brand identity refers to the conceptual and strategic core of a brand. It encompasses the values, positioning, personality, and tone of voice through which an organization seeks to define how it is perceived. He does not work on the physical form because it is not directly visible; somewhat, it is shaped by the meanings, expectations, and emotional associations. In this sense, brand identity represents the internal logic and intention behind the brand, with the main answers to the questions of what the brand stands for and how it differentiates itself from competitors.

Visual identity would then be a visible form of brand identity. It shapes conceptually into visible elements such as logos, color systems, typography, and graphic structures. Visual identity enables recognition and consistency across communication channels, functioning as the most immediate interface between the brand and its audience.

"A brand identity program encompasses a unique visual language that expresses itself across all applications. The challenge is to design the right balance between flexibility and consistency." (Wheeler, 2018)

It is clear that branding is an overarching process that brings brand identity and visual identity together. It includes strategic planning, market positioning, and the coordinated deployment of both conceptual and visual elements across multiple touchpoints.

Branding is therefore not a single outcome, but an ongoing practice that shapes and builds a brand communication and maintains over time (Spelova, 2024).

Now that we have established an instance between these terms, we can discuss branding as a system. If brand identity defines meaning, and visual identity defines form, branding defines the mechanisms through which meaning and form are aligned, reproduced, and adapted.

"The goal in creating a brand identity is not surface consistency, but inner coherence." (Balkind, cited in Wheeler, 2018)

In contemporary practice, everything becomes more fluid. The logic of applications varies across different formats and communication situations, in which identity remains recognizable but can shift. Visual identity is no longer a collection of elements but a visual language. Language remains identifiable even when the sentences change.

In branding, language can create connections among proportions, hierarchy, contrast, rhythm, tone, and a structure of constraints. Wheeler uses this comparison to suggest that visual language today works in harmony across applications while maintaining a balance between flexibility and consistency. He emphasizes coherence over superficial uniformity, suggesting that strong systems need not be rigid (Wheeler, 2018).

This is extremely important in institutional settings, where the design is created by several individuals rather than a single designer, and the identity is a shared framework that everyone who participates in its creation must use, interpret, and apply over time. All rules defined during creation serve as a basis for easier, faster, and safer decision-making, while still allowing for interpretation.

Johnson also supports this principle from a practical branding perspective, where modern identity systems should not be treated as closed projects. As the organization grows, teams change, the media, the environment, implementation possibilities, and technology evolve, the institution's visual presentation should keep pace. In that case, this kind of flexibility is not a weakness but a strategic decision, provided it establishes rules and a strong structure that account for variations at the beginning of its creation or rebranding (Johnson, 2016). If this flexibility is not established at the very beginning, variation occurs as inconsistency. On the other hand, if it is well-tuned and designed, variation is seen as growth.

Many designers have already recognized and supported such concepts, long before their actual application. From theoretical practice, that design not only produces solutions but also programs solutions. Gerstner's concept of program became a framework that defines elements, rules, and relationships by which outcomes are generated. Even then, one could hear that identity is not a fixed form, but a set of decisions that shape the entire field of valid forms.

The program determines the limits within which variation is possible, ensuring that differences remain connected (Gerstner, 1964). This argument is a clear example of why coherence and variation are not opposites, but co-dependent qualities in contemporary identity systems.

Of course, thinking about such flexible systems at the time focused less on practice, where, without proper technologies, it was difficult to achieve, at least, continuity of production. We are witnessing significant technological changes, and, as Manovich says, platforms will inevitably change, but the fundamental condition of increased scale, production, and circulation remains (Manovich, 2020). Predicting the program's future is not possible, at least it is not available to designers, who create in and for the present, following trends. That's why it is important to create the space that remains operable across contexts that cannot be fully anticipated in advance. Building such a system means making room for extension without having to redesign it each time communication formats shift.

In this sense, system design, essential for institutions, becomes necessary. And if the practice of not producing a single artifact has been established for a long time, it is now raised to an even more serious level, and a family of related outputs, different in appearance but connected through an underlying logic, is called (Manovich, 2015). It is not only to define how something should look once, but to define how it can continue to look like itself while adapting to change.

Manzini's interpretation of design as a transition from creating objects to creating circumstances and frameworks helps situate it within the broader field of design.

The designer increasingly creates frameworks that enable others to generate outcomes cohesively when design is dispersed across networks, organisations, and long-term processes (Manzini, 2015). In institutional branding, this is not abstract: it describes the everyday reality of identities that must remain consistent without the need for constant centralised control. Therefore, a system is more than just a beautiful structure. It is a practice model that facilitates continuity over time, assigns accountability, and organizes decisions.

Taken together, these viewpoints describe current branding as an organised balance of stability and accessibility.

Coherence can be achieved by defining relationships and constraints that allow for variation rather than by limiting identity to a single form. This perspective sets the stage for the next section, which introduces tools and automation as ways to implement, scale, and maintain branding systems rather than as external additions.

"Contemporary design increasingly focuses on creating frameworks that enable others to act, rather than designing finished objects." (Manzini, 2015)

In practice, maintaining system-based identities often involves tools that translate conceptual rules into repeatable actions. Such tools do not determine design decisions, nor are they a necessary condition for successful branding, but they can support the application of complex systems across scale. In this way, while procedures are used in daily operations, automation may seem more like an optional mechanism for managing consistency and variation than an intrinsic value of branding.

1.2 Automation and Tools in Branding Practice

After realising that branding is not a fixed set of visual components but rather a system of relationships and applications, we move on to its practical application. This implementation is increasingly being done through tools used for contemporary branding. These tools make systems easier to implement, distribute, and maintain at various levels by operationalising existing rules rather than replacing design decisions.

Therefore, automation in branding should not be viewed as a singular technological advancement or as an essential or intrinsically beneficial development. Instead, it appears as an optional extension of systems thinking: a means of facilitating accessibility, efficiency, and consistency when identities must operate across various teams, contexts, and formats. In this way, tools serve as a valuable conduit between branding concepts and their real-world implementation.

1.2.1 Automation as an Extension of Branding Systems

As branding systems spread across different media, formats, and organizational contexts, their practical applications increasingly depend on tools. Their "abstract" rules are translated by the tool until the desired visual concept is achieved. When this happens, certain parts of the design can be automated to the extent that they are repeated in the same or similar frames (Fig 1.2). An example of this is the increasing presence of templates and generative programs. Automation in contemporary branding does not necessarily emerge as an external technological force imposed on design practice; it can already be seen as a continuation of system-based thinking already present in identity design.

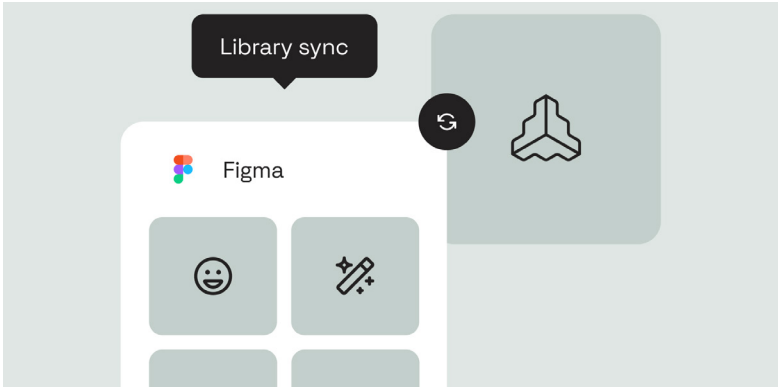


Fig. 1.2 Example of automation workflows in contemporary branding systems. Source: Frontify, Brand Automation Guide, accessed 21 February 2026.

Relationships, limitations, and hierarchies can be operationalised with tools that facilitate their long-term, consistent application after they have been precisely defined.

It is crucial to stress that branding automation is not a new development and is not the same as artificial intelligence. Branding structures adopted standardised templates, modular layouts, and predetermined rules to ensure a consistent appearance long before computational methods entered the design discourse.

An early example of automation is brand manuals, which encode decisions so they can be reused without renegotiation (Rebelo et al., 2022). In contemporary branding, automation should be viewed as the redistribution of design judgment across tools, systems, and procedures rather than its replacement. Software environments that govern the creation, use, and maintenance of identities are increasingly integrated with design choices.

Design judgement is redistributed rather than replaced by automation. By automating certain brand management services that require intricate data processing and content creation, AI-driven tools such as Large Language Models further expand this redistribution (Wiestner et al., 2024). However, rather than being the focus of implementation or analysis, such technologies are only mentioned in the present research to contextualise a more general change in practice.

Practically speaking, automation becomes important when large-scale branding systems are required. Large institutions and organisations frequently lack centralised day-to-day control and operate across teams, departments, and communication channels. To function as a long-term system rather than a completed project, branding must endure change rather than resist it. In such circumstances, tools are there to preserve coherence as various actors apply identities over time rather than to impose rigidity (Gregersen & Johansen, 2021).

The before mentioned logic, in which design does not consist in producing a single solution but in defining elements, rules, and relationships that allow multiple solutions to be generated, can help us better understand automation. In this case, automation speeds up work because following predefined rules prevents it from being reinterpreted each time manually. A program is not an autonomous machine, but a responsibility-bearing structure. Automation thus extends the designer's intention rather than replacing it (Lyons et al., 2021).

This extension often takes the form of tools that encode identity logic into interfaces. Online template-based systems, asset libraries, and modular layout frameworks allow designers to work within established constraints while adapting outputs to specific contexts (McKenna & Silbey, 2023).

In everyday practice, designers should be aware that interfaces are not neutral, but they shape how systems are understood and used. Some decisions are made explicit, while others remain hidden (Manovich, 2015). This becomes more complicated when discussing systemic branding. Institutions must balance stability and adaptability while operating in media environments where specific platforms and technologies inevitably change (Manovich, 2020).

Institutional brands should consider their own automation tools rather than depending solely on existing options available to everyone, also to maintain a coherent identity in unpredictable future contexts. This means that automated systems must be carefully constructed to support informed choices rather than obscure them.(Light et al., 2017)

Of course, automation is not the goal of branding. It is there to support the system's clarity, scalability, and consistency, but only when used correctly. Used incorrectly, it can compromise and limit a visual identity. Institutional branding systems need to skillfully navigate and balance the use of such tools in practice to maintain control and flexibility.

1.2.2 Existing Tools and Practices in Contemporary Branding

Today's branding draws on a diverse mix of tools, from hands-on design apps to semi-automated platforms working at various levels of complexity. Though they vary in focus, goals, and tech savvy, together they show how brand systems are built, managed, and scaled in the real world.

As one of the most used branding workflows sit everyday design tools like Figma and Adobe Creative Suite, essentials for crafting identities, giving designers precise control over visuals, layouts, and typography. But lately they too have relied on system design, shared libraries, components, design tokens, and variables turn identities into flexible rule sets rather than just one visual. This keeps things consistent across teams and projects, though it still demands hands-on judgment when applying them.

When it comes to large teams and institutions, they choose to complement standard design software with online brand portals to

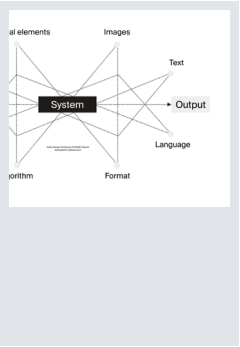


Fig. 1.3 Example of rule-based partial automation, component-driven interface (Framer platform).

make manipulation and editing easier. Those platforms are usually already equipped with visual styles and examples, enabling non-designers to have sufficient aesthetic control over what they do. They increase consistency and speed up the process, but they're just vaults for sharing ready-made resources, not for creating new ones. Some platforms that enable partial automation are no-code and low-code platforms such as Webflow and Framer, which offer slightly clearer forms of rule-based design in branding workflows (Fig 1.3).

They allow a more fluid visual appearance, especially on the web, with component structures, layout constraints, responsive rules, and content management systems that generate multiple outputs from a shared logic. Here, designers define the system that the layout will later be adapted to, enabling non-designers to make easy changes. It should be noted that these platforms and tools focus less on branding systems.

Canva is an example where automation is obvious, especially in its template-driven format, through features like Brand Kits and automated resizing.

These systems enable the rapid production of branded materials within fixed templates that can be reused across different contexts. However, we are not talking about true branding automation here, because the Canva platform is available to everyone and prioritizes efficiency. Still, it does not enable control over authorship and brand coherence.

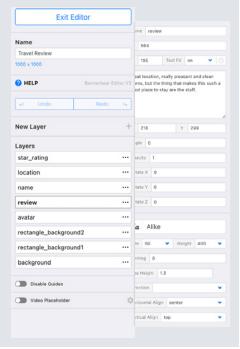


Fig. 1.4 Rule-based layout system (Bannerbear).

Recently, rule-based and generative tools have emerged that avoid the template-based frameworks of platforms like Canva. Platforms like Bannerbear create visual representations from data and simple layout rules. Designers can set relationships, constraints, and behaviors, skipping the final setup and allowing for much greater opportunities for meaningful variation (Fig 1.4). Yes, this is mostly for marketing or content creation, not for direct branding, but it hints at a larger shift: seeing visual identity as a playground of options, not a locked box of resources.

Most branding tools out there have a significant problem: they address only one part of the branding process at a time. Some help with organizing assets, others with ensuring layouts look

consistent, or even with generating content. But very few of them treat branding as a whole, connected system, thinking about the brand's core idea, how it's made, how well it works, and how it changes over time. Because of this, automation often feels tacked on, like an extra step, instead of being built right into how we design brand identities from the start.

While automation is great for making branded materials quickly and on a large scale, it only really works well in branding when people are still in charge and guiding it. Studies show that automated tools can help maintain a brand's look and feel, but only if humans make the big decisions. Automation should be seen as something that allows our branding systems, not as an independent creator (Clarke & Joffe, 2025).

Even though automated systems can generate many options, they don't truly understand concepts like meaning, storytelling, or cultural context, we still need human judgment. (Ortí et al., 2025). It just changes how we share design responsibilities across different tools and steps, rather than eliminating them.

If we use automation without clear rules and ways to assess its effectiveness, we risk making the design process less clear rather than helping it.

1.3 Politecnico di Milano Visual Identity as Institutional Case

The visual identity of Politecnico di Milano is an excellent example of institutional branding built as a structured system. It relies on a small set of carefully chosen elements that keep everything recognizable across departments, media, and time.

Typography follows this structured approach. Primary (Manrope, Arial) and secondary (The Frank Ruhl Libre, Georgia) font families are modern, neutral, and legible, working well in print, on digital displays, for signage, and in publications. They visibly represent clarity, scalability, and lasting power over trendy style. The rules governing their hierarchy in titles, spacing, repetition,

and proportions are well-defined. These typography rules must facilitate repetition and transmission. When different people communicate over time, adhering to the rules keeps the meaning and structure intact, even if the content changes.

Primary typeface:
Manrope
Secondary typeface:
Frank Ruhl Libre

Ag Una vita senza ricerca non è degna per l'uomo di essere vissuta. **Platone**, Apologia di Socrate

Ag Una vita senza ricerca non è degna per l'uomo di essere vissuta. **Platone**, Apologia di Socrate

Blue Heritage
#102C53
100 60 0 75
539 C
16 44 83

Color selection is one of the clearest considerations in systemic brand building. The core institutional blue, pure white and black the exact shades chosen for print and digital content – are like a solid foundation. They convey continuity and trust, exactly what people expect from a university. A narrow color palette eliminates ensures that items from different departments match perfectly every time.

Secondary colors come in, but as controlled additions. They are used mainly to highlight departments, events, or specific messages, without disturbing the overall feel. Here again, the systemic idea is emphasized where things can change, but only within certain limits. The meaning of a color comes from its position in the system's hierarchy. Tertiary colors are there to complement the secondary, but with clearly defined uses for departments; all shades are derived from blue with different dominants to refer back to the institutional primary color.

Templates and layout settings further demonstrate this system's thinking. Headings, subheadings, grids, images, margins, and alignments are all designed to be applied to different events, contributions, and digital media (Fig 1.5). Visible margins strengthen institutional enforcement and are a good example of design that follows the rules, giving a consistent results while adapting to specific needs.

Fig. 1.5 Template example developed according to the Politecnico di Milano brand guidelines.

Geometric and radial shapes are the best foundation for developing a role-based system. They organize space and guide the eye's flow.

Following strict rules of circularity, they exhibit a strong form of generative emergence (Fig 1.6). By properly establishing the rules, they become parameters that can generate new forms, which was later part of this research.

Responsive design, flexible typography, and adjustable graphics show that the identity is built for change, especially in digital spaces. This proves that it is aimed at long-term flexibility, not rigid short-term control. Communication has the ability to grow, shapes create space for patterns, the grid gives the possibility of escalating typography.

This setting is very important for institution, with many departments and channels. Relying on parameters, hierarchies, repetitions, the visual identity of the Politecnicno di Milano is ideal for introducing computational design.

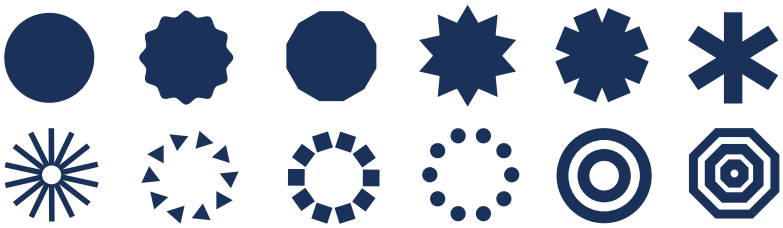


Fig. 1.6 Partial subset of geometric elements in Politecnico di Milano visual identity.

1.4 Institutional Constraints as a Design Problem

A big network of systems is essential for institutional branding. Commercial brands usually have product lines that require short-term campaigns. Still, universities, government institutions, and cultural organizations must maintain a consistent visual identity over the long term, for diverse audiences, and across multiple units. In this environment, brandin is always an ongoing process. In practice, institutions face the usual branding challenges. First, branding systems must be flexible and used across all media, departments, and situations but also remain controlled to ensure everything looks recognizable and official. Second, fast production is important and this is often not done by the

main central design team. Third, consistency must be maintained even when non-designers are making decisions and technology is constantly changing.

These challenges reveal a fundamental tension in institutional branding: balancing stability with flexibility. As McKenna and Silby (2022) show, design emphasizes continuous processes over one main product, prioritizing coherence over matching styles. Coherence does not come from the repetition of identical objects, but from the systematic alignment of rules, relationships, and constraints.

Design tools are key here because they respect the rules of identity and allow for controlled variation without constant manual adjustments. Traditional brand guides help codify guidelines, but they still rely on human judgment. As systems scale, this approach becomes fragile.

Manovich (2015; 2020) argues that software environments actively shape the way systems are interpreted, applied, and transformed. When interfaces, templates, portals, and content management systems become part of branding systems, certain decisions are enforced while others become invisible. This shift alters the role of design and designers by establishing a dependency on technical infrastructures.

From this perspective, design thinking is limited and fails to account for the fact that institutions do not simply implement identities but also manage them over time. This leads to systems that need to recognize variability while maintaining autonomy.

This chapter focuses on how contemporary branding, especially in institutional contexts, already operates as a system of rules, relationships, and constraints. The increased need for scalability, adaptability, and continuity reveals a structural gap between existing design tools and the operational realities of branding practice. Bridging this gap does not require replacing designers; instead, it requires rethinking how design systems are built, codified, and managed.

02

Computational Design in Branding Practice

When branding is considered a system rather than a fixed visual identity, it opens the door to computational design. Identity is shaped by relationships, rules, and structures rather than pre-established forms, which is also very important. Design then shifts from creating individual elements to building systems that allow for variety while remaining consistent. In this way, computational design is not just a new technology, but a natural step in the system-based approach already found in modern branding.

Computational design does not replace traditional design principles. Instead, it makes them clear, practical, and easier to scale. Rules that used to be understood but not written down, such as those found in brand manuals, templates, or a designer's experience, can now be set as parameters, limits, and steps. So, computational design does not bring an entirely new way of thinking, but offers tools to handle complexity, variation, and repetition more intentionally as communication needs grow.

Computational design is often used to achieve automation, but it is not the same as automation or artificial intelligence. These technologies can be part of how computational design is put into practice. The main focus of computational design is on how design decisions are made, carried out, and improved. It is a mindset that values processes over outcomes, systems over objects, and relationships over individual parts. This approach works well for branding, especially in organizations where brand identities must remain consistent over time, across different media, and among various teams. (Tierney et al., 2022)

In branding, computational design lets designers explore multiple visual options rather than settling on a single solution.

Computational systems create rules, and designers use parameters to test out various related styles. Constraints help ensure the

results remain clear and recognizable. With this method, designers focus less on controlling every detail and more on setting up the rules that shape the final outcome. (Gunagama, 2018)

Putting design rules into systems means choices have to be clear: limits must be set, connections explained, and ways to judge results decided. These choices do not go away; they just move to a different place (Douglas et al., 2021). Because of this, computational systems shift design decisions away from removal, making the designer the creator of the system rather than just someone who makes single items.

This chapter examines how computational design is used in branding, focusing on the designer's perspective. Instead of looking at purely automatic or AI-based solutions, it explores computational design as an organized approach to creating, managing, and evaluating differences within identity systems.

The discussion ranges from rule-based, adjustable design ideas to systems with interfaces that let designers work and control computer processes. The chapter ends by showing where the shape generator developed in this research fits, not as a replacement for design work, but as a tool grounded in human judgment, feedback, and organizational rules.

2.1 Rule-Based Design

Rule-based and parametric principles are usually part of standard design systems. Often, these principles work behind the scenes, helping to create consistency, variation, and scalability in different settings. Moving from designing single objects to creating systems of rules is a significant shift in how we think about and make forms (McCormack et al., 2004).

Typography is a typical example of rule-based design. Typefaces use sets of rules to define characteristics such as size, shape, spacing, and pattern. Even though each letter looks different, they remain consistent because there is a clear logic behind the system that all letters follow. Variable fonts build on this idea by letting

designers adjust features such as thickness, width, and size. So, typographic systems work more like flexible spaces than fixed sets of styles.

This approach is similar to branding systems that require controlled variety rather than just a few set options (Parikh et al., 2025; Reas et al., 2010). Variable and parametric typography has recently been talked about as a way to shape future visual identity systems. One such example is Prototypo, where designers manipulate typographic parameters through interactive controls instead of choosing fixed styles (Fig 2.1). By offering a range of options rather than fixed styles, these systems let designers explore different looks rather than just picking from set choices.

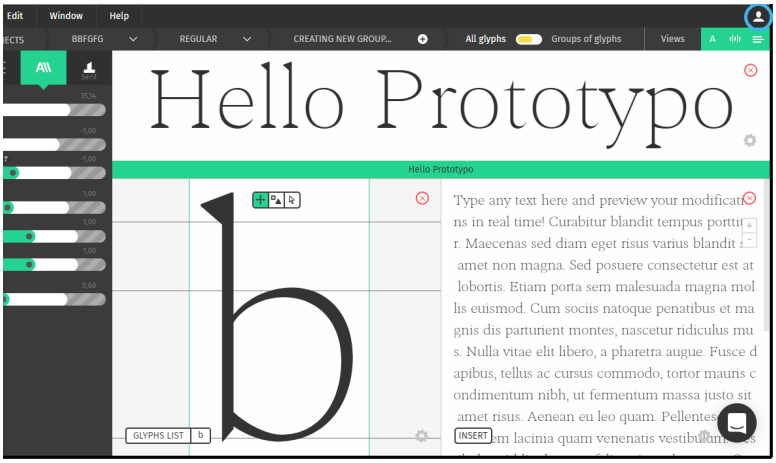


Fig. 2.1 Interface of the Prototypo parametric typography tool demonstrating adjustable typographic parameters within a rule-based design environment.

Recent research shows that an adjustable design helps with gradual changes rather than sudden style shifts, making it a good fit for identity systems that need to evolve over time (Bauer & Frere-Jones, 2020).

Grid systems are a good example of rule-based thinking in design. Editorial and institutional projects often use modular grids to set margins, column widths, alignment, and ordering. Designers work within these rules instead of creating new layouts each time. Some

grid systems go further by allowing the structure to change based on content, format, or medium (Liao & Hu, 2023). This approach keeps posters, publications, and digital interfaces consistent while remaining flexible. As Gerstner explained, consistency arises from stable relationships that drive variation, not just from repeating the same thing (Gerstner, 1964).

Brand color systems are often flexible. Instead of using the same set of colors everywhere, many identity systems set rules for how colors work together, such as assigning main and supporting roles, limiting contrast, and specifying when to use certain colors.

In institutions, these systems allow departments or sub-brands to change their colors in a controlled way while maintaining a shared logic. In this approach, color is a flexible part of the system, not a fixed feature, so identity is built through structure rather than sameness. Morrison (Meirelles) points out that visual coherence relies more on the clarity of the system than on individual elements (Meirelles, 2013).

Recent design research shows that rule-based layout systems are not just technical tools, but also ways of thinking that help guide creative ideas. Studies on how people use computers show that designers rely on clear limits to make decision-making easier and to help them focus when working on complex projects. Instead of stifling creativity, these well-made rules help people explore, providing a steady base for new ideas (Owen & Peckham, 2021). This way of thinking sees limits as the foundation of creative freedom.

Generative approaches appear in graphic identity systems composed of simple shapes. Projects that use circles, grids, building blocks, or branching patterns establish rules such as turning, resizing, repeating, or moving shapes. These rules create groups of related visuals rather than just single designs. In this approach, variation is expected and planned, not a mistake. This way of thinking aligns with McCormack's view of generative design, where the final look emerges from the process rather than manual work alone.

Platforms like Processing and p5.js have made these ideas real, turning abstract concepts into practical experiments. First made for creative coding and experimentation, these tools show how rules and settings shape what is created (Fig. 2.2).

Fig 2.2 Rule-based visualization generated in Processing, demonstrating dynamic parameter variation through interactive mouse input.

Designers set limits and watch as changes spread through the system. This process of adjusting and watching moves the focus from making perfect still images to understanding how things work and relate. While these are not branding tools themselves, they show a way of thinking that works well in identity design. Outside of testing and experiments, systems that use set rules and adjustable settings are increasingly used in professional design through tools that require little or no coding. These tools turn step-by-step instructions into easy-to-use controls, so designers can change settings without writing code.

Research on visual programming tools highlights how they help make systems' workings clearer and easier to adjust, helping designers connect computer logic with their own creative ideas (Myers et al., 2020).

Parametric logic is also important for presenting information and graphics based on data, where things like position, size, color, and how crowded something looks are carefully matched to the data. Morrison says that good visual systems depend on using these matches and relationships, not just on making things look different for decoration. While branding does not show data in the same way, it faces a similar challenge: staying consistent across

different content and situations (Meirelles, 2013). A common idea in these examples is that rule-based and parametric systems move creative work from making single pieces to setting up the rules, limits, and links that shape everything.

As Cross (2018) points out, today's design skill is about understanding the whole set of problems, not just making quick solutions. Real creativity is about seeing change ahead of time and building systems that stay clear and connected as they grow.

These practices show that rule-based and parametric design are not just computer ideas, but real strategies used in today's visual culture.

They matter for branding because they help support planned variation, growth, and long-term consistency, which are especially important for big organizations. This idea leads into the next section, which examines design programs and generative thinking as ways of working that precede and shape computer use. This way of working focuses on making rules and logical connections, which is at the heart of parametric design thinking. It lets designers explore more solutions than traditional methods. This view aligns with the ideas of generative algorithmic modeling, which uses both relationship-based and productive modeling to automatically explore design options using predefined design rules rather than doing everything by hand (Delerel & Bayram, 2025).

It should not be left out that generative and parametric design highlight that rules based systems require clear direction from the person who sets them configured. Even if the results look automatic or vary each time, the person who sets the rules, limits, and ways to judge them is still responsible. Current critiques of generative AI in design keep highlighting this, saying that real variety comes from careful system design, not from the system working on its own (Engawi et al., 2022)

When we see rule-based and parametric design as ways to create new visual ideas rather than just automating, generative thinking becomes a real design approach rather than just a new technology trend. This difference will be important in the next discussion about design programs.

2.2 Design Programmes and Generative Thinking

The idea of designing through programmes predated the existence of computational environments. Long before digital tools, forward-thinking people like Karl Gerstner saw design as moving from producing a single result to building systems that allow many different outcomes. In this way, design is less about giving a final answer and more about setting up the right conditions for solutions to appear. A programme is not a machine, but a way of thinking: a plan that shows what could happen, instead of forcing one single way.

This way of thinking is especially important for branding systems. Consistency is kept not by repeating the same shapes, but by following the same basic idea. Differences are not mistakes, but planned and expected results of the system. Gerstner's difference between a programme and a recipe is important here: a programme gives a range of possible answers, while a recipe gives only one set result. In branding, this difference helps identity systems stay steady while changing to fit new situations, formats, and ways of communicating.

Generative design, in this sense, does not rely on certain technologies or computer programs. It is not the same as automation, and it does not always mean artificial intelligence. Instead, it is a way of setting up design problems so that many different results can be made within clear rules. Computer tools later made this process clear and usable, but the basic idea of generative thinking came before computers (Gaier, 2020). This human skill to create generative systems supports the making of unique, changing events that show the designer's main vision. This difference is important to avoid the wrong idea that generative design takes away authorship. Instead, the designer is still responsible for setting the plan, its limits, and how it all fits together (Hansen et al., 2021).

When looking at branding that uses automated and rule-based methods, certain common ways of creating variety stand out. Even though these approaches are rarely given names in the projects, they keep appearing in different studios and situations. Variety usually comes from specific actions that change parts, how they

connect, or how they act over time, not just random changes. Many studios now use systems that let them control how things change across brands, campaigns, and platforms. In these situations, the designer's job is not just to create single items but to build systems that others can use, understand, and add to over time.

One group of approaches can be seen in studios that use rule-based or automated systems as a key part of brand identity. Studios like Studio Dumbar/Dept, Studio Moniker, and FIELD.IO often create identity systems where change is built into the visual style itself. FIELD.IO recently did a Instagram Global Gradient Toolkit (2024) that demonstrates how rule-based visual systems generate coherent yet variable outputs (Fig. 2.3). Here, movement, shapes, and set rules create groups of designs that still look related. The identity is not just one fixed look but a flexible system that can show up in many ways.

In these approaches, change usually happens by following set rules or actions. Shapes, movement, and how things interact are the main ways to create different looks, leading to groups of results instead of just one design. Change can happen by turning or resizing shapes, or by using effects like pulling together, pushing apart, or moving with invisible forces, which let the identity change over time while still staying connected.

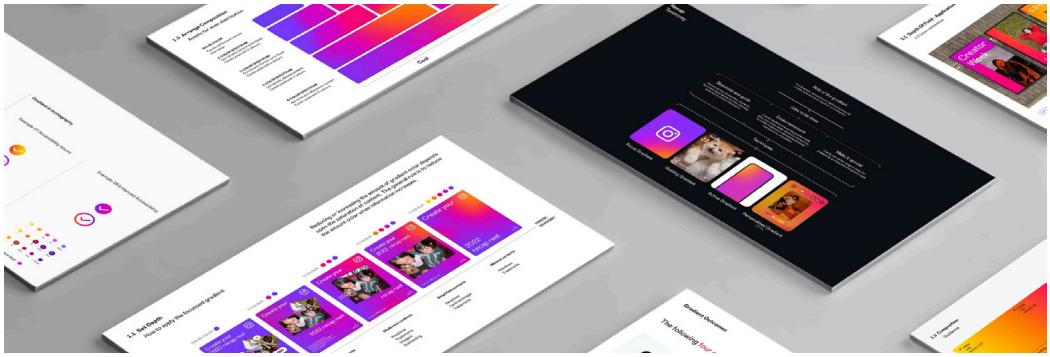


Fig. 2.3 Instagram Global Gradient Toolkit, FIELD.IO (2024). Source: FIELD.IO official website, accessed 22 February 2026.

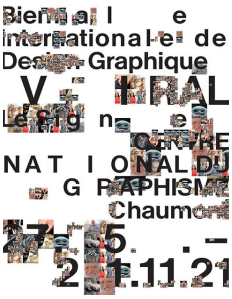


Figure 2.4 Chaumont Biennale 2021 identity system by DIA (2021). Source: Diversions Magazine (accessed 22 February 2026).

Fig. 2.5 Experimental typographic poster by Bureau Borsche, illustrating process-driven deformation and transformation as a design method within experimental identity practices. Source: Another Graphic (accessed 22 February 2026)

A second group of practices sees change as a creative tool, using it to share ideas rather than to create strict systems. Studios like DIA Studio and Kurppa Hosk demonstrate this with flexible layouts, creative font use, and repeating graphic elements that can move and change across different media. An example of this structured yet non-computational flexibility can be seen in DIA's identity for the 2021 Chaumont Biennale, where typographic fragmentation and modular composition allow variation while maintaining visual coherence (Fig. 2.4).

These methods may not use computers, but they work by establishing connections and rules that enable playful rearrangement. By mixing and matching steady parts like modular fonts, grid-based layouts, and signature graphics, identities can fit any situation while keeping their main look. Here, variation comes from skillfully reusing familiar parts, not from computer-made changes.

A third category includes practices that create custom tools or scripts for specific projects, often blending branding, computer-generated art, and experimental design. Studios such as Vrints-Kolsteren, Ultragramme, and Bureau Borsche sometimes use custom computer processes to create visual material. Experimental practices that use custom tools or scripts often try more extreme types of changes. These include bending, changing shape, or other changes that greatly alter the original form. One example of this transformation-driven approach is seen in the experimental typography of Bureau Borsche studio. Here, identity changes arise from altering the structure of forms rather than following set systems (Fig. 2.5).

A common pattern emerges through these practices; generative and program-based methods are increasingly used, but they are rarely set up as clear, reusable systems. These applications are good for content driven visualization and variation that is clearly planned, where experimentation is possible.

Although generative thinking is prevalent in idea development, its emphasis on novelty and experimentation with different aesthetics is often unsuitable for organizations and institutions that require a consistent, enduring design identity. Systems might produce interesting results, but not having clear ways to use or

get feedback from these systems also makes them harder for non-technical designers to use (Su et al., 2024).

From a branding perspective, this raises an important question: how can programmatic and generative principles be translated into tools that support everyday design practice without obscuring decision-making or removing human judgment? If design programmes define a field of possibilities, then branding tools must enable designers to navigate this field consciously rather than automate outcomes invisibly.

These issues are especially important in organizations, where brand identities need to endure, work across departments, and be used by many people. In this way, current methods show both what generative thinking can do for branding and where it falls short. This challenge is the main idea behind the review of current computer-based branding tools in the next section and the shape generator developed in this research.

2.3 GUI-Based Design Systems

A graphical user interface (GUI) is a way for users to interact with a system by clicking on elements they can see, instead of typing commands or code. Buttons, sliders, input fields, menus, and other controls turn the system's functions into things you can see and use. Unlike command-line or script-based interfaces, GUIs let users work with complex systems by seeing what happens in real time (Zhang et al., 2024).

In design work, GUIs are very important for connecting ideas with what you see on the screen. They let you use, modify, and observe how a system works without changing the code directly. This matters a lot in computer-based and rule-based design, where the system operates by following rules and settings rather than using predefined shapes (Ferreira & Leitão, 2015). As computer-based methods are increasingly used in branding and identity design, the interface becomes an important space where design choices are made. A GUI does not just show a system; it also affects how people

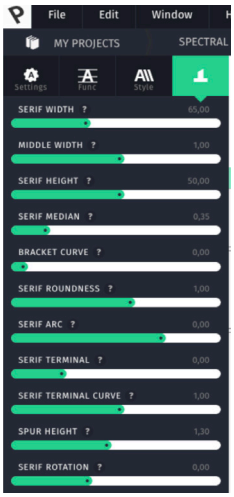


Fig. 2.6 Graphical user interface exposing typographic parameters as adjustable controls in a rule-based system. Source: Prototypo interface documentation, (accessed 22 February 2026).

understand and use it. The choices for settings, the limits for those settings, and how the controls are arranged decide how designers try out different options and how much control they have over the results (Morrison, 2017).

Unlike systems that rely solely on scripts, GUI-based systems let designers see, test, and compare changes right away. Settings can be changed step by step, and you can see the results immediately (Fig. 2.6). This helps designers work in ways that align with common design methods such as sketching, building models, and comparing visuals (Zhang et al., 2024).

From this point of view, systems with graphical interfaces do not eliminate complexity but merely move it around. Choices that would normally be hidden in the code are now shown as settings you can change. The interface becomes a layer that helps explain the system, making its logic partly visible and open to change (Whitelaw, 2015). Shifting control in this way affects who creates and collaborates on projects. Script-based tools like Processing or custom-coded systems are very flexible, but they usually put control in the hands of those who write and manage the code. In contrast, GUI-based systems let multiple designers work together without needing to deal directly with the underlying code.

In organizational branding, this difference is especially important. Branding systems need to be used repeatedly by different people and across different departments. GUI-based systems let the basic rules stay the same while allowing designers to adjust results for different situations. They act as a middle step between strict templates and systems that run completely on their own, balancing human decision-making with ease of growth and consistency.

This way of looking at GUI-based design systems connects to the review of current computer-based branding tools. The next section examines how current platforms apply these ideas in practice and where there are still problems with automation, openness, and designer control.

2.4 Existing Computational Branding Tools

Current digital branding tools mostly fall into two main groups: tools that require coding and tools that use ready-made templates and extensive automation. Both types help with making changes and working quickly, but neither one really treats branding as a clear, flexible system that can grow, be checked, and managed over time.

Fully code-based tools like Processing and p5.js let designers set up exact rules and connections for how things are made. These tools offer the greatest freedom and creative options, allowing people to build complex visual systems through code (Fig. 2.7). Because of this, they are often used in experimental branding, brands based on data, and moving graphics. But since you need to know how to program, not everyone can use them. The rules for making changes are hidden in the code, so designers who do not code find it hard to understand, change, or reuse the system without help from the person who made it. So, even though these tools are powerful, they are not often used in big organizations where many people need to work together and keep things running over time.

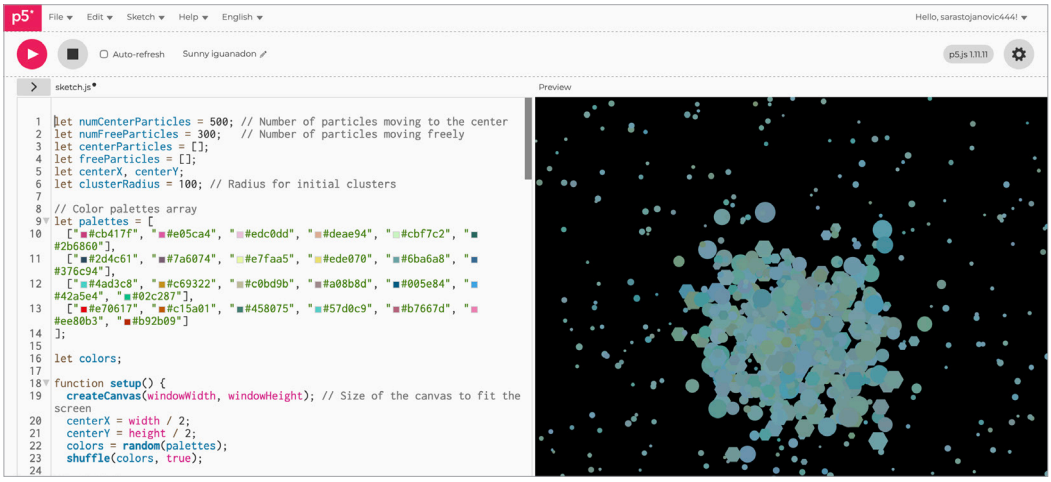


Fig. 2.7 Code-based generative system implemented in p5.js, showing rule definition (left) and resulting visual output (right).

On the other side are branding tools that focus on automating and making things easy to use. These include design systems with templates, brand websites, and tools that use AI to help create content quickly with set layouts and automatic choices.

These tools make work faster and more consistent, but they often hide how the brand system works. Designers mostly work with the finished results instead of the rules behind them, which can make the results stiff or too similar. In these cases, automation is often just added on top of how things are already done, instead of being built into the main way the brand is designed.

Between these two types, a smaller but growing group of mixed digital tools is emerging. These tools try to connect the power of programming with easy use by mixing rule-based actions with visual interfaces. For example, Mechanic is a design tool that lets designers set up their own functions, settings, and step-by-step actions while using a clear visual interface. Instead of just making fixed templates, Mechanic allows changes through rules that can be seen and adjusted, like repeating, changing, or setting conditions (Fig. 2.8).

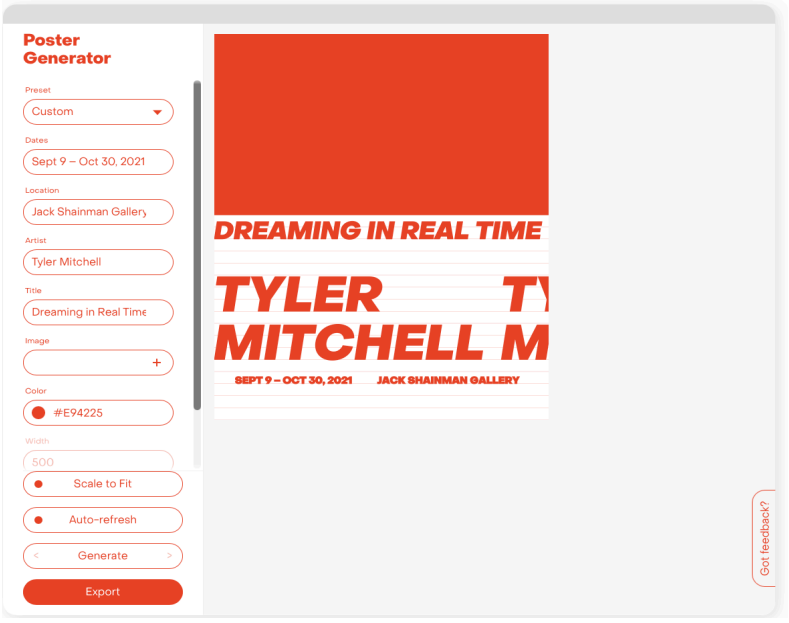


Fig. 2.8 Rule-based visual interface for parameter-driven poster generation in Mechanic. Source: Mechanic design tool interface, (accessed 22 February 2026).

This mixed approach makes it easier to use digital design by integrating the system rules into the interface. Designers do not have to write full code, but they can still see how changes are made. The interface acts as a go-between for the rules and the final visuals, making it clearer how the system works compared to fully automatic tools.

Other tools also demonstrate how rules work through visual or adjustable interfaces. For example, Grasshopper for Rhino lets designers build complex rule-based systems by connecting visual blocks. While these tools are very creative, they often require a lot of technical skill and are mostly used in building design or fabrication, not in branding. Observable notebooks and other visual coding tools also mix clear rules with visual results, but they are usually used for exploring and studying data, not for brand design.

Even though these mixed tools have promise, they are still limited. Most are made for trying things out, making single pieces, or short-term projects, not for full branding systems. They rarely have ways to check, improve, or manage work across teams and departments. Also, many are closed systems, which makes it hard to add new features or keep using them for long in large organizations.

In all these types of tools, there is a common problem: most tools only handle one part of the branding process. Some organize files, others make layouts, and others create content automatically, but very few bring together the brand's rules, make changes, and check results in one clear system. Because of this, digital branding is still split up. The lack of a middle option that is both flexible and clear, programmable and easy to use, shows there is a big gap in how things are done now.

This gap is especially clear in large organizations, where brand identities need to remain consistent over time, across different locations, and with many people involved. In these cases, neither hidden automation nor tools that need coding really help people work together or use the system for a long time. The next section explains how the shape generator developed in this research addresses the missing middle layer, offering a system that uses clear rules, human judgment, and controlled changes in a way that is easy to use.

2.5 Positioning the Shape Generator

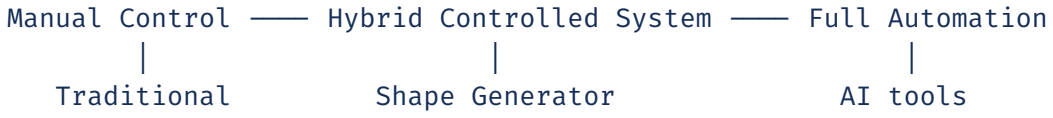
Looking at computer-based methods, creative thinking, and current branding tools shows an ongoing challenge in today's design work. Manual design lets designers have a lot of control, make choices, and respond to context, but it is hard to use at scale and maintain consistency in large organizations. Automated and AI-based tools are fast and can generate many options, but they often obscure how they work, leaving the designer more of a chooser or overseer. There is still a largely unexplored area for tools that help designers explore options in a structured way without turning over all design decisions to automation.

This research puts the shape generator between manual and automated design. Instead of replacing manual work or becoming a fully automatic system, the generator is meant to help designers explore options within clear rules and limits. Its goal is not to make finished designs, but to let designers try out different shapes, compare them, and make better choices before picking a final result.

Unlike fully automated branding tools, the shape generator does not try to hide its workings behind predetermined results. Instead, it makes the rules and settings clear and easy to change. This openness is important in organizations, where design choices need to be clear, responsible, and match the brand's identity.

By letting designers work directly with the parts that create different options, the generator keeps the designer's role while letting the system handle some of the work. The generator is not a template tool or an AI that generates on its own. It does not automate meaning, story, or style. Instead, it automates aspects such as sizes, patterns, and rules already set in the branding system. In this way, the generator builds on planned thinking, by turning a design plan into a working system without forcing one specific outcome.

The position of the shape generator within current design approaches can be summarized as follows:



This approach also distinguishes the generator from the other computer tools discussed earlier. Some tools focus on speed with strict automation, while others enable creative experimentation through custom scripts. The shape generator, however, is made for long-term use in organizations. Its design is purposely limited to keep things clear and consistent, not just open-ended. The range of variation is controlled, so all results fit within the brand's look.

The generator is intended to serve as both a research tool and a design aid. It lets designers create, compare, and choose shapes repeatedly, supporting ongoing exploration rather than just one-time results. This repeated process will be important in later research stages, where feedback and improvement are built into the system. In this way, the generator makes checking and judging part of the design process, not just something done at the end.

For organizations, this approach answers real needs mentioned earlier: being flexible without losing unity, staying consistent without being too strict, and growing without losing control. The shape generator does not try to fix these issues solely through automation, but rather by carefully designing a tool that balances system rules with human choices. It gives designers a safe space to try out different options without risking the brand's look and feel.

At the same time, the generator is not meant to be finished or work for every situation. It is a basic system that can be improved later. Ideas such as making it adaptable, learnable, or adjustable to user preferences will be explored in future research. This choice shows the main idea of the thesis: computer-based branding tools should be built step by step, with people testing and understanding the system before adding features such as machine learning or letting it operate on its own.

By placing the shape generator between manual and automated design, this research suggests a new way for computational tools to help, not replace, designers. The generator acts as a bridge to the design plan, allowing designers to explore options in a structured way while maintaining control. This lays the groundwork for the next chapter, which will examine how the generator is designed and used.

PART II

System

03

Early Exploration and System Foundations

The previous chapter explained computational design as a way of thinking in terms of systems, rules, and methods for creating things, rather than just using specific tools or technologies. It showed how using programming and adjustable methods helps designers handle complex visuals, deal with differences, and create groups of related designs. However, these methods only make sense when used in real design work. This chapter shifts from theory to practice. It explores how computational thinking applies to real situations, workplace limits, and teamwork in design. Rather than presenting a finished system, the chapter traces the early steps of creating the shape generator, focusing on the decisions, changes, and ideas that influenced its development. The focus is not on new technology, but on what drives design: how certain needs, constraints, and teamwork led to moving from fixed templates to a more flexible, programmatic approach to creating visual elements. By describing this early stage, the chapter presents the generator as a response to real branding needs and growing design knowledge, rather than just a computer experiment.

3.1 Motivation for a Shape Generator

The idea to develop a shape generator came from real, practical needs found while working on an internship with the Politecnico di Milano Communication Office and its design team. The goal was to experiment with technology or add automation if it was possible and needed. But at the end, the project addressed daily challenges such as maintaining visual consistency, scalability, and adaptability across a large branding system. It was a challenge to describe these needs in technical terms.

At first, the project was focused on using templates to automate design tasks. The main idea was to turn existing brand guidelines, especially for print and some web uses, into a flexible set of digital templates for social media. This would let departments create their own branded visuals while following set rules for layout, typography, and color.

The early approach was inspired by tools like Mechanic, which use templates with placeholders. It allows users to click controls and edit text within a predefined structure, while adding flexibility by being open to coding.

The concept was to move away from a manual design workflow by converting design rules into tools. At that time, automation was seen as a way to reduce repetitive work and help departments create content faster. The focus was on working efficiently and maintaining consistency, not on creating new designs. But as talks with the Communication Office continued, it became clear that using templates was not the priority, because the brand manual had relied on well-defined rules for templates.

These conversations revealed another need: to expand and organize the use of supporting graphic elements in the Politecnico di Milano's visual identity. The focus turned to the circular and radial shapes already used in the brand. Although these shapes were well-defined, they were often used in limited, repetitive ways. Designers wanted a more flexible way to create, modify, and extend these forms while maintaining the brand's identity.

This change shifted the project's focus. Instead of just creating templates to hold content, the team began generating the visual

elements themselves. The idea of a small object generator came up, designed as a tool with a few settings to create different circular shapes. These shapes could then be used as building blocks for patterns, backgrounds, or layouts in various types of communication.

After further exploration, it was realized that these shapes could be built using clear rules. Radial forms, for example, are defined by settings like size, angle, repetition, and proportion. This showed that the system should not just copy existing shapes, but should include the rules for making them. Later, this approach grew to cover not only circles but also other shapes with straight sides, making the system more flexible and expressive. Its purpose and use gradually changed as a result of talking, testing, and thinking. What started as a way to automate templates became a way to turn brand ideas into a system that could create new designs.

This process shows that the generator was never meant to take over design decisions or work on its own. Instead, it was built to support designers by making brand rules clear, visible, and easy to use. The goal was not to create finished designs automatically, but to provide a safe space for exploring different forms while maintaining consistency with the brand's identity.

In this way, the shape generator was motivated by both practical needs and systems thinking. It answers real demands like scalability, consistency, and efficiency across departments, but also asks a bigger question: how can a branding system become programmable without losing its meaning or flexibility? The shape generator is not just a tech fix added to design, but a tool shaped by real practice and ongoing testing. Each new question or consideration increases the potential for additional features; therefore, this discussion will focus on the aspects presented in the following chapters. This generator exemplifies the flexibility and functionality of a strategically developed brand such as Politecnico, demonstrating its capacity to adapt and evolve in response to emerging applications and future requirements.

3.2 Shape Vocabulary Development

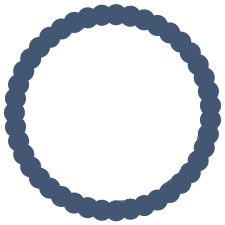


Fig. 3.1 Radial base form used as structural unit in the shape generator system.

The creation of a set of shapes marks a transition from experimentation to the establishment of a coherent system. At this stage, the generator functions as a design guide informed by deliberate decisions, rather than merely facilitating random changes. The selection of shapes constitutes a critical design step that determines how the system communicates ideas and meaning.

This approach aligns with design research, which highlights the importance of decision-making and reflection in the design process. As Schön (1983) explains, designers gain knowledge by reflecting on materials, constraints, and outcomes. The created shapes are not fixed; they change and improve through repeated testing and refinement. Each shape within the system reflects the institution's identity and incorporates insights from initial experiments.

Instead of using a wide range of geometric forms, the set was developed to reflect the visual identity of Politecnico di Milano. This identity is characterized by geometric shapes, particularly circles and radial designs, which serve as key visual elements in both print and digital media. These shapes serve beyond decoration, organizing space, establishing rhythm, and maintaining visual consistency across various applications.

Radial shapes serve as the main building blocks of the design. They offer stability and flexibility, making them recognizable no matter the size, angle, spacing, or repetition. Radial shapes work well in systems that follow simple rules, making them suitable for brand designs that need to be consistent and adaptable (Fig 3.1).

As the design evolved, we added polygons to the shapes. This decision was based on the needs of the design, not simply because we could. While circles provide smoothness and balance, polygons bring direction, detail, and distinct sections. These elements help differentiate messages or themes, all while staying true to the brand's core style.

However, adding polygonal shapes required careful thought about their visual impact. Sharp-edged polygons can look rigid or harsh when overused in branding. To address this, we incorporated rounded corners. This change focuses on how shapes behave rather than how they look, aligning with ideas discussed in computational

and generative design literature (Reas et al., 2010). Technical research on how to build shapes supported this approach. For example, Gorilla Sun's algorithm for polygons with rounded corners shows how curves can be made using arcs, keeping the shape smooth while allowing control over the corners. While the generator in this research does not copy any one method, these examples helped show that visual softness can be built into the system itself, not just added later. This supports the idea that design qualities should be part of the system from the start.

The set of shapes is kept small on purpose. This is a design choice to keep the meaning clear and prevent the style from drifting, not because of technical limits. Design research shows that strong systems come from working within clear limits, not from endless options. By limiting the options, the system ensures that changes still fit the same identity, not just random variations.

Importantly, the shapes in the set are not seen as finished objects but as flexible setups. Each shape is defined by rules and limits that determine how it can change. This aligns with the idea that design is about setting the right conditions for shapes to appear, not just making each shape directly. In this way, the designer's job is to define the range of options, not just produce single examples. Even if the radial shapes were planned from the beginning so that they fit into the branding with the given parameters, it was difficult to achieve consistency.

Polygonal shapes, which by the number of sides form circular forms and fit into the branding, were difficult to harmonize with all the parameters of the radial elements; new ones had to be thought of, and because of this, there were more than a few deviations in the satisfactory appearance of the final results that would be applied as visual material. So the division into radial and polygonal shapes alone was not enough, but their fit had to be perfected and determined.

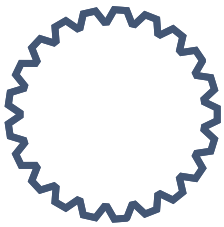


Fig. 3.2 Polygonal element variant illustrating improved extension of the geometric vocabulary.

In this way, building the set of shapes is more about designing the whole system than just making forms. It creates the basic language the generator uses, making sure every result, no matter how different, still makes sense and matches the identity (Fig 3.2). This set serves as the basis for all the rules and choices that follow, letting the generator serve as a way to explore and adapt designs in a controlled way.

3.3 Initial Experiments

Early experiments with the system began before there was a clear idea of what the generator should be, or whether it would be a generator at all, or a strict rules interface. At this point, the goal was to explore the possible shapes suggested by the Politecnico di Milano identity and to investigate whether simple rules could already generate a useful variety of shapes. Everything started using the p5.js online platform which allowed “small” creative coding and real time checking, because the canvas that was set as the initial segment was automatically updated.

Existing objects were analyzed to identify regularities. All objects have the same maximum dimensions and can be divided from a circle into columns that, together, form a circular shape. The number of columns is variable; fewer columns result in greater width, while more columns produce narrower segments. Additionally, these objects may feature curved corners and can form circles from smaller basic shapes by repeating and rotating small circles, squares, and triangles. Then, the polygonal part was introduced, to which functions were added, initially the same as for the linear forms, then changed to provide relevant results of the basic forms from which the polygonal forms arise: the star, gear, and the polygon (Fig 3.3).

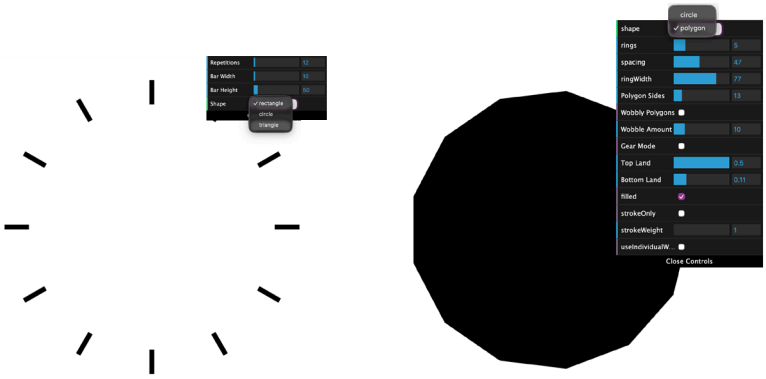


Fig. 3.3 Early generative experiments in p5.js showing radial base structures and parameter-controlled polygonal extensions during initial system development.

In this research, randomness is used as a design tool rather than merely a computer effect. When used early on, random generation helps show everything a set of rules can do: it produces surprising results, uncovers unusual situations, and reveals the system's limits. At the same time, these tests show why using only randomness does not work well for branding.

A brand system needs to be easy to recognize, steady, and work the same way in different situations and for different people. Early tests with complete randomness, then, did two things: showed what is possible, but also why structure, limits, and clear settings are needed.

3.3.1 Radial Element Sketch

```
let gui;
const OUTER_
RADIUS = 300;
```

One of the first test versions was a simple p5.js sketch with a dat.GUI control panel, using a set outer size of 300. This choice set a key idea for later versions: the output always fits within a set boundary, but the details inside can change. So, the system explores variation within a fixed space instead of changing the overall size.

The sketch creates a ring of repeated simple shapes (rectangles, ellipses, triangles) arranged in a circle. This setup works as a radial repetition engine, using a few parameters to build a structured visual result by rotating, moving, and repeating the shapes.

The fixed frame was therefore an outer radius constraint, so that the object maintained the same position and dimensions. It was a circle that functioned with the constraints that nothing could exceed the maximum circumference, references, i.e., that all elements were referenced to the same radius, and a built-in rule that things would get out of hand.

```
function draw-
FixedBars(cx,
cy, count,
barW, innerR,
outerR) {
let angleStep
= TWO_PI /
count;
```

After that, by analyzing existing university branding objects, the basic parameters of the GUI were determined.

Repetitions control how many pieces are placed around the ring. In simple terms, it sets the angle between each piece:

```
angleStep = TWO_PI / count
```

In terms of design, this setting works like a "density" control. Fewer repetitions make the design look simple and emblematic; more repetitions make it look more like a texture or a border. This one setting changes the look from icon-like to decorative to pattern-like.

```
let maxBarW =
(TWO_PI * out-
erR) / count
* 3.0; barW =
constrain(barW,
1, maxBarW);
```

Bar width (barWidth) controls how wide each piece is. Bar width cannot go past a certain maximum:

```
maxBarW = (TWO_PI * outerR) / count * 3.0
```

This is an early example of the system controlling itself: it does not let the bar width get too large, but sets the maximum based on the same shape that determines the number of repetitions. This helps prevent pieces from overlapping or looking too stretched, but it also shows a main problem with early versions: the limit is based on math, not on what looks good or makes sense. It keeps the design from breaking, but it does not always yield useful results.

```
let barLength =
outerR - barH;
for (let i =
0; i < count;
i++) {
 ....
let midR = barH
+ barLength / 2
```

Bar height (barHeight) controls the radial thickness of each repeated primitive. Here, bar height also indirectly defines where the element sits, because position uses the midpoint radius:

```
midR = outerR + barH / 2
```

Changing bar height changes both the size of the piece and where it sits on the ring. This link is visually strong, it helps keep the pieces lined up with the outer edge (they usually stay attached to the circle's border). But it also shows an early design problem: a single setting controls both size and placement, making it harder to adjust later.

Shape gives a possibility to choose between rectangle, circle, and triangle shapes. This quickly changes the look while keeping the main structure the same. It also acts as an early "vocabulary" test: the same rule can use different basic shapes, but the way it looks changes a lot.

At this stage, the system used repetition, rotation, and local orientation. Each element was placed on a circle and rotated so it aligned with its radial direction.

For position element center is placed at radius midR on angle angle. Orientation is happening as rotatation by angle, so the element faces outward/tangentially depending on primitive orientation. Using push() and pop() keeps local transformations separate.

```
push();
translate(x, y);
 ...
pop();
}
```

Each unit acts as a module that can be repeated and rotated. This modular approach is important later, as it allows the system to grow from single modules to more complex designs while maintaining a consistent structure. The initial sketch does not try to solve a branding project; instead, it checks whether a limited parameter set can create rich visual variation and which parameters strongly influence perception (count, thickness, proportion), where problems show up, like too much repetition, bars that don't match in size, or confusing links between bar height and where things are placed.

This stage also highlights an important point: shapes can look organized with simple geometry, but only at a basic level. The results are tidy, but they do not meet the brand's needs. Right now, the system can create different shapes, but parameter variation could exceed intended limits, producing unstable or unusable results (Fig. 3.4). The development process showed that rules shape what happens, having some control is important alongside randomness, and that adding more choices in a smart way can make things much better.

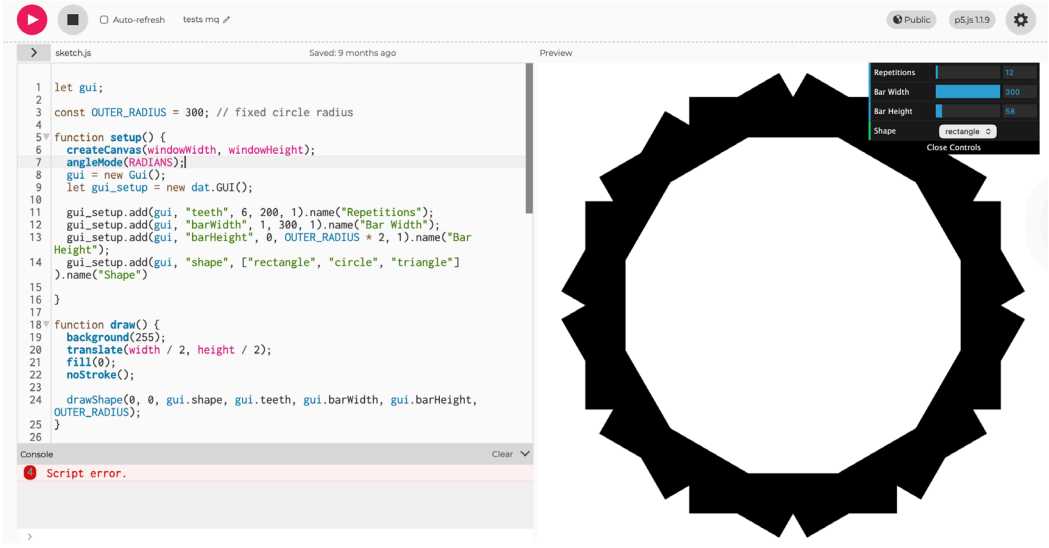


Fig. 3. 4 Early test result demonstrating loss of control when parameters exceeded predefined structural boundaries, highlighting the necessity of constraint-based system design.

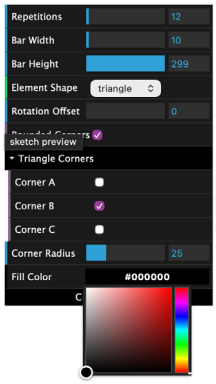


Fig. 3.5 Early GUI settings with added rotation offset and rounding controls.

```
gui_setup.
add(gui, “ele-
mentRota-
tion”, 0, 360,
1).name(“Rota-
tion Offset”);
gui_setup.
addColor(gui,
"fillColor").
name("Fill
Color");
gui_setup.
add(gui,
“cornerRa-
dius”, 0, 100,
1).name(“Corner
Radius”);
```

Rules act like a guide, shaping what happens even when they are simple. Systems that create things on their own do not remove choices; instead, they put them into rules and limits. There is a need for a better program strategy, not just expanding parameters.

The next steps focused on improving the GUI layers and making them easier to use, with clear data-driven controls. Now, instead of only using code, you can see and use controls to change design settings right away. The main structure stays the same, but you can now adjust more settings in the same way as before. Along with the basic options from the first version, the sketch now adds:

`rotationOffset (elementRotation)`: a setting that turns all the elements around the circle, so you can rotate the whole design without changing how the pieces are arranged.

`fillColor`: an early way to let the generator make designs, but you choose the color with a color picker instead of using brand colors added later.

`roundedCorners and cornerRadius`: a switch and a slider that let you make the shapes less sharp and smoother, instead of only having hard edges. At first, the curves were very strong, and the edges were clear, with little flexibility.

One important thing about this version is that the interface is not always the same. The sketch uses code (updateCornerFolders()) to show or hide different controls based on what shape you are working with and if rounded corners are turned on. For example, the controls for rectangle corners (“Top Left”, “Top Right”, “Bottom Right”, “Bottom Left”) only appear when you are working with rectangles and roundedCorners is enabled. The same goes for triangle corner controls. This is a small but important step toward making the tool smarter, because the interface now matches what the generator does internally, rather than showing every option all the time.

For triangles, rounding is done by a special function (roundedTriangleArc) that draws curves at the corners instead of using standard shapes. The code determines the side lengths, corner angles, and where to place the curves to create a rounded triangle. You can also pick which corners to round—corners A, B, and C can

each be rounded or left sharp. This means the generator can now make many different kinds of triangles, not just one simple type.

Even though the sketch is still about trying out new ideas, this way of rounding shapes already shows a common problem in making shapes with code: settings like "corner radius" cannot just be of any value, because they depend on the shape's size and angles. Rounding has to match the side lengths and angles, so the system has to set limits and rules inside.

3.3.2 Polygon introduction through "Gear Mode" and concentric shapes experiment

As the early circular sketches settled into patterns of repetition and round layouts, another approach brought in rings within rings as a second way to organize things. This experiment is based on the idea that using rings and circles is not only easy for computers but also fits well with the simple, geometric style already used in the Politecnico di Milano identity, where circles and repeated shapes are common design features.

The prototype uses a menu-based system that lets you switch between two main shape types: circles and polygons, adding a special "`Gear Mode`" for polygons.

Gear mode was introduced as an early extension of the polygonal shape vocabulary, allowing polygon edges to alternate between inner and outer radius, forming tooth-like structures. Instead of just making simple polygons, the gear mode builds each "tooth" by switching between points close to the center and points farther out:
`Npoints` defines the number of teeth.
`innerR` and `outerR` define the tooth depth.
`topLand` and `bottomLand` define how much of each tooth is flattened at the top and bottom.

An optional useCurve mode draws the shape with `curveVertex`, producing a smoother, more organic contour. This changes how polygons can vary: instead of just changing how many sides

they have, you can now change the shape of each tooth. It's not an ideal way to create variable polygons, but it served as a starting point for elaboration and reflection.

Alongside gear mode, the sketch includes useCurve + wobbleAmount for "wobbly polygons," where vertex radii are perturbed with a sinusoidal offset and rendered with curveVertex. This is valuable as an exploration tool because it quickly reveals how small perturbations change perceived tone, more rigid/technical versus more playful/organic. Still, in institutional identity contexts, such noise-based deformation typically requires careful justification and stricter bounds, otherwise it risks drifting away from a controlled visual language [Fig 3.6].

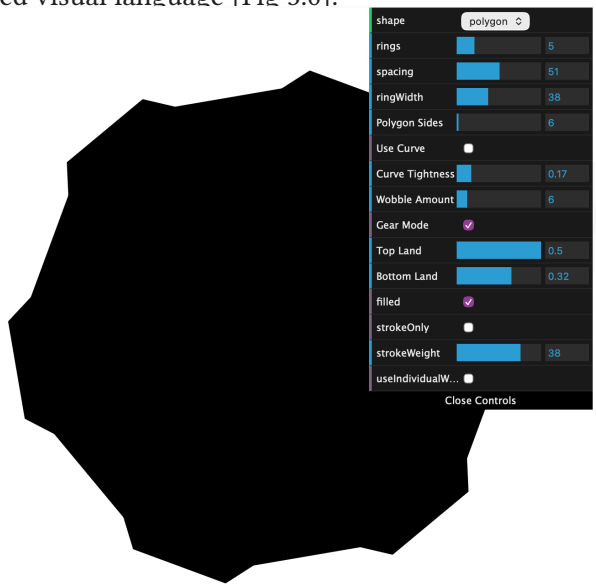


Fig. 3.6 Early polygonal exploration illustrating parameter instability prior to the introduction of structural constraints (gear mode, top/bottom land settings).

After the first steps in building polygonal shapes, concentric logic was introduced. The system is built around rings, or layers, that are nested inside each other.

Each layer is separated by a specified amount of space, and each can have the same width or its own width if that option is enabled. The sketch at this step does not use a strict rule for how the rings

fit together; instead, it just adds a bit to the radius for each new ring: For circles, the radius increases by the width plus the space between rings, which creates a clear stack of rings. The drawing can show the width as the outline thickness or as a filled band, depending on the settings.

For polygons, however, the concentric behavior is more approximate. The radius for each ring is calculated as: `r = i * params.spacing + 40`, which means the spacing parameter drives the overall scale, while width becomes primarily a stroke attribute (unless "filled" is used). In this early form, the concentric polygon rings do not yet behave as true geometric offsets of the polygon boundary, but as repeated redrawings of polygons at increasing radius [Fig 3.7].

It is important to note this rough way of making rings: it shows that making rings with code is not always the same as making rings that look right in design. To make rings that look good, it should be decided how size, outline, space, and the edges fit together, especially when polygons have corners and can overlap.

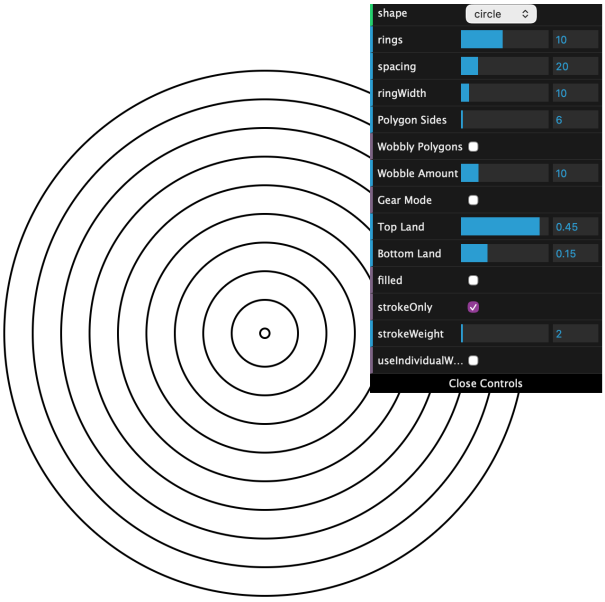


Fig. 3.7 Early concentric radial experiment exploring parameter-based repetition and spacing constraints within the initial generator prototype.

However, the sketch also shows a common early problem in creating shapes with code: the gear system does not yet follow the rules to keep everything balanced. For example, some settings for topLand and bottomLand can make the teeth look odd, and the way ring size and tooth depth interact is not yet consistent across all rings. These are not mistakes at this stage, but signs that making a set of shapes needs both ways to create and ways to control them. At this point, it is just used as a test: it shows which kinds of shape changes are easy to see and which turn into random differences.

These first experiments show that randomness lets us see different visual outcomes and discover surprising new options, but it also shows that order, meaning, or unity is required for brand identities. Pure randomness gives us options, but it does not help us make thoughtful choices.

As the experiments continued, it became clear that the system needed places where changes could be directed rather than just observed. This change is a move from just exploring to clearly shaping the results: from making everything to deciding what can change, how it can change, and what the limits are. At this point, settings become the main way to bring design choices back into the system.

3.4 Parameter Exploration

Even though during the initial settings, the generator lacked sufficient definition to establish later, more specific rules, the system began to use clear settings to add more control over time. This stage was a shift from just trying things out randomly to using more organized changes. These settings were added slowly, in response to problems with how clear, useful, or suitable the results were for branding. In early sketches, settings were kept simple and directly linked to basic shapes. This lets the designer see how each change affects the overall look and structure before adding more complex links between settings.

3.4.1 Primary and Secondary Parameters in Early Exploration

The first group of settings focused on circular structures and repeating patterns, as these matched the round,geometric style of the Politecnico di Milano visual identity.

Repetition count (teeth / repetitions) ranged typically from 6 to 200. This parameter controls the number of elements distributed around the circle. Lower values produce clearly distinguishable, emblematic forms, while higher values approach dense textures or patterns. Early experiments revealed that extremely high values reduce legibility and clarity of identity, suggesting the need for upper constraints in later versions.

Element width (barWidth) ranged from very small to a maximum size based on the outer circle and the number of repeats. Element width affects how heavy and regular the design looks. If not limited, this setting can cause shapes to overlap and look messy, so rules were added to keep the width in check based on the outer circle's size.

Element height or length (barHeight) ranged from 0 to the edge of the outer circle. This setting controls how far shapes reach toward or away from the center. At the smallest or largest values, shapes either shrink to the edge or take over the whole design, showing how size shifts the look's balance.

Shape type (rectangle, circle, triangle) is a setting that switches between shapes rather than changing smoothly.

Adding different shape types early on lets the designer compare how each shape behaves, revealing differences in balance, symmetry, and the design's overall feel.

Together, these settings made up the basic structure of the system. Changing them led to significant differences in how everything looked, so they were key to maintaining the design style.

As testing continued, more settings were added to explore different looks without altering the main structure.

Rotation offset ranged from 0 to 360 degrees. This setting turns the shapes around the center, adding a bit of movement while keeping the repeating pattern.

Corner rounding (boolean + radius) ranged from sharp corners

to a radius constrained by element dimensions. Rounding the corners showed that even small shape changes could really change the feel of the design, making it look less stiff and more natural.

Compared to the main structure settings, these changes only affected small parts of the design. They changed the style without making it hard to recognize, showing that some settings are more important than others.

Some settings have a bigger effect on how the system works. Main structure settings control how clear and recognizable the design is, while style settings only change small details. This aligns with the idea in design theory that large settings shape the overall design, while small settings adjust the details.

Seeing which settings mattered most was important for the next steps. It helped decide which settings users could change, which should stay the same, and which needed limits.

This stage of testing settings showed that a generator is not just a blank set of choices. Every setting shows a design decision, and every range is a choice about what is a good result. So, the focus moved from adding more settings to improving how they work together and setting limits.

These lessons led straight into the next stage, where the generator was set up as a system made of parts with a clear structure, rules, and a way to use it. Chapter 4 explains how the early settings were turned into a well-organized, flexible design tool.

A. MAIN STRUCTURE (identity-defining)	B. STYLE MODIFIERS (appearance tuning)
• Repetition count (teeth / repetitions) • Element width (barWidth) • Element height (barHeight) • Shape type	• Rotation offset • Corner rounding (toggle + radius constraint)

Fig. 3. 8 Parameter hierarchy used in early generator explorations.

3.5 Iterative Refinement

As shown in the examples of early coding and option setup, building the generator did not follow a straight path from idea to finished system. It changed, started over, and grew through a repeated process of testing, watching, changing, and reworking. Each round added new limits, removed unnecessary choices, or changed how options worked together. This process was needed to turn the generator from a rough idea into an organized design system.

Early versions gave us interesting results and let us try new shapes, but as the code got more detailed, more branding issues appeared. We learned from these challenges by limiting which settings could be changed, choosing new starting values, and ensuring that some settings worked well together. This approach helped us improve the process and make thoughtful design choices.

In each round of building the code system, we changed options, looked for new results, and watched for patterns. Some outcomes always looked better, made more sense, or fit the institution, while others felt random. These observations shaped our later design choices. The generator became both a creative tool and a way to learn. By repeating the process, we saw which parts needed more control and which could stay flexible.

Unsuccessful or disappointing results were not considered mistakes, but as useful clues. They helped us discover new ideas about form, order, and balance that we had not noticed before. For example, having too many options made the results messy and off track, while too many limits made things dull and repetitive.

Both situations showed the need to balance freedom and control. Some settings were not needed or changed things too much, and some combinations caused problems. To fix this, some settings were removed, others were combined, and others were controlled by how they worked together rather than by fixed numbers. In this way, failure became a resource for design rather than a problem. Some tests with the shape generator did not work for branding, even if they were correct or looked good. These tests helped set the system's limits and improve its design rules.

4

Building the Shape Generator

While the previous chapter looked at early testing, experimenting with different settings, and gradually building a set of shapes, this chapter shows how these ideas were brought together into a clear system. It describes the move from trying things out to building a well-organized, practical shape generator. The shape generator is seen not as a final product, but as a system that grows through repeated thinking, testing, and changes. It is not just one program, but a design system made of different parts that work together: how shapes are made, the range of settings, and the user interface. This way of thinking moves from just trying things out to building an organized system, where design choices are built into both the tool's look and structure. The idea was that design systems need to balance flexibility and control. If everything is automated, it can mask how decisions are made, but if the system is too strict, it cannot adapt to different needs. So, the generator is designed as a tool for exploration, not just for creating finished designs. Its main job is to help designers move through a set visual space, making it clear how changing settings affects the results. From an interaction design perspective, this chapter examines the generator as something people use and interact with, not just as a technical tool. The interface is not just a simple layer, but an active link between how the system works and the choices people make. As interaction design experts point out, good systems let users explore, change things, and reflect on the results, rather than just accepting what the computer gives them (Sharp et al., 2019). The chapter starts by describing how the system is set up, showing the differences between how it works, how it looks, and how people use it. Next, it looks closely at how shapes are made, then reviews the controls, the ways results can be saved or shared, and examples of how it can be used. Finally, it discusses the lim-

"Designing is a reflective conversation with the situation." (Schön, 1983)

its of using fixed rules and prepares for the review and changes discussed in the next chapters.

Design systems are not only technical structures, but interactive artifacts shaped by human decision-making. Interaction design theory sees systems as tools that help people explore, get feedback, and learn, rather than just doing one set job. Sharp, Preece, and Rogers say that good interactive systems show how they work inside, so users can understand how their actions change the results. In generative design, this means that settings, limits, and how the system acts should be clear to designers. In the shape generator, this idea manifests as clear controls for settings and instant visual feedback, ensuring the designer is actively involved rather than just watching the computer.

4.1 System Architecture

The shape generator is designed as a modular system rather than a monolithic tool. Its setup is carefully planned so that each part has its own job and can change independently while still working together as a whole. This way of organizing things is not just about technology but also about how the system is built and improved. At a high level, the system is organized around three primary layers: generative logic, visual output and user interface. The generative logic layer defines the rules, settings, and shape relationships that govern how shapes are generated. This part presents the system's main ideas: how shapes are built, how settings work together, and what limits the visual area. This logic is designed to work on its own, independent of how things look or how users interact with it.

By keeping the rules separate from how things are displayed, the system lets you change the main logic without having to quickly redo the interface or how results are shown. The visual output layer is responsible for displaying the shapes generated by the system's rules. It turns the rules and numbers into shapes you can see. This part does not decide what shapes to make, but only

how they look. Because of this, you can change things like line style, color fills, or file types without changing the main rules for making shapes. The interface layer (GUI) connects the system to the designer. Instead of showing all the complex rules, the interface only shows certain settings as controls you can adjust. This makes the system easier to use and understand. The interface does not make choices for you; it provides a space to try things out, so designers can work with the system by changing controls rather than editing code directly.

This way of separating parts is important for making changes and adding new features over time. As the system changed, settings were added, changed, or removed, shapes were improved, and new actions were tested. Having separate parts made it possible to make these changes step by step without breaking the whole system. Changing and improving the system is built into the generator's design, not something added from the outside.

From a reflective practice perspective, the system functions as a material site for thinking through design decisions. Schön describes design as a conversation with the situation, where practitioners reflect on the consequences of their actions and adjust accordingly (Schön, 1983). In this context, the shape generator is not a finished artifact but a medium for reflection-in-action. Each architectural adjustment responds to insights gained through use, testing, and evaluation, reinforcing the system's role as a dynamic and evolving construct.

Also, keeping these parts separate helps the system grow and handle more uses in the future. By keeping clear lines between the rules, the way things are shown, and how users interact, the system can be updated for new types of shapes, ways to show results, or ways to check results without needing to rebuild everything. This setup matches the goal of making a system that is flexible, easy to understand, and ready for use in organizations, not just as a one-time experiment.

This separation in the system also makes it clear which decisions are being made. Choices about shape, size, patterns, and rules are handled by the part of the system that creates the basic design options. Choices about color, line thickness, filling, and

how things look are handled by the part that controls appearance and affects how the design is seen, not its structure. Choices about which settings are available, how they are organized, and how they can be changed are handled by the part that designers use to interact with the system. By making these differences clear, the system keeps important design decisions from getting lost in technical details.

Instead of just making the system faster or more automatic, it is built so that it stays easy to understand as it changes. Each part can be examined, changed, or discussed on its own, allowing you to see where each design choice came from. This makes it easier to review and reflect on the work later, helping people understand not just what the system produces, but also why it produces those things.

By avoiding hard-coded visual outcomes or automated decision-making, the system ensures that control remains with the designer. The generator proposes possibilities and parameters act as points of negotiation between the designer and the system, allowing intentional exploration without surrendering authorship. Computation here serves as an extension of design thinking.

This way of building the system makes it easy to add new features over time. As the project grew, new shapes, settings, and ways to interact could be added without rebuilding the entire system. For example, changing the generator from making only round shapes to also making shapes with straight sides mostly meant changing the part that generates the designs, while the user controls and appearance only needed small updates. This shows that keeping the system parts separate helps it grow and change easily.

Finally, the system architecture provides a foundation for subsequent evaluation and adaptation. Because parameters are explicitly defined and isolated within the logic layer, they can be logged, analyzed, and compared across iterations. This makes the system suitable for structured testing and user evaluation, which becomes essential in subsequent chapters.

4.2 Shape Generation Logic

The shape generator works by following a set of rules, using a limited set of shape rules and settings to make each result. Instead of just drawing shapes, the system creates a controlled workspace. A set of outer-edge marks defines the space for making shapes, while settings inside the system control how often shapes repeat, how big they are, how they turn, and how they act. The system is set up so that changes are made by adjusting things like the number of shapes and their sizes, angles, widths, and heights, rather than drawing by hand. This way of working lets the system create detailed patterns and forms by adjusting settings, so that new designs emerge when you change the inputs (Trautmann & Piros, 2020).

4.2.1 Structure and Initial Parameters

The shape generator's structure sets the system's size limits and the shared information used throughout all drawing and interaction steps. This first part of the code sets up fixed values, global variables, and data setups that form the base for interface outputs.

The constant `OUTER_RADIUS` sets the largest size the system can use. All shapes, whether round or with straight sides, must fit inside a circle of this size, ensuring a stable and predictable visual frame and comparability between generated outputs; no matter how complex things get or how the settings change, all shapes must stay within this limit.

```
const
OUTER_RADIUS =
300;
const
GUIDE_COLOR =
“rgba(0, 0, 0,
0.1)”;
```

`GUIDE_COLOR` is a semi-transparent outline color used only for testing and visual help, like drawing the outer circle or boxes for reference. Keeping guide colors separate from the real shape colors ensures that these helpers do not get mixed into the final look of the shapes.riables store references to the rendering surface (canvas) and the interface model (gui).

4.2.2 Setup Function, Color System and Palette Structure

The `setup()` function starts the system and sets up its basic rules. `createCanvas(windowWidth, windowHeight, SVG)` makes a drawing area that can be scaled to any size. Using SVG instead of a regular image means everything looks sharp at any size and can be easily exported, printed, or edited later.
`angleMode(RADIANS)` standardizes all angular calculations. Since trigonometric functions in JavaScript operate in radians, this avoids repeated conversions and reduces the risk of angular inconsistencies.
`rectMode(CENTER)` simplifies `rectMode(CENTER)`, making it easier to place shapes by letting you position them around their centers instead of a corner, which is especially helpful for circular layouts. `()` initializes the graphical user interface, while `hideGUI()` hides the interface container visually without disabling its logic. This separation allows the GUI to function as a control layer even when it is not visible.

Finally, `randomizeSettings()` creates a starting setup, and `noLoop()` stops the system from redrawing over and over. The system only redraws when settings change, making it a tool for exploring designs rather than for making animations.

```
/*———————————————— SETUP ———————————————— */
Function setup()

  createCanvas(SVG)        →          scalable output
  angleMode(RADIANS)       →       consistent geometry
  rectMode(CENTER)         →        radial positioning
  createGUI()              →         control interface
  hideGUI()                →  logic active / UI hidden
  randomizeSettings()      →            initial state
```

```
Brand Color
System

Primary
Secondary
Tertiary
   ↓
GUI Controls
category
selectedColor
opacity %
   ↓
System Logic
find hex
convert to p5
color
setAlpha()
   ↓
Rendering
fill() /
stroke()
```

The color system is sorted into groups and each group has named colors. This setup does a few things: it copies how brands organize their colors by purpose, lets the interface fill in drop-down menus based on the chosen group, and keeps color choices separate from shape logic. Colors are named, not by their color codes. This makes the system easier to read, change, and update later without breaking how shapes are made.

Even though color might seem like just a visual detail, in this system it is handled as part of the main setup because it must adhere to the official color choices. Colors are split into main, department, and extra categories, based on the branding guide: main colors are the institution's main colors, department colors are for different departments, and extra colors are added to department palettes. The code picks a color by:

```
gui.category (Primary / Secondary / Tertiary)
gui.selectedColor (named token such as “Blue Heritage”)
gui.opacity (0–100%)
```

The color is chosen by finding its hex value in colorCategories, then turning it into a p5 color and setting its transparency. The setAlpha() function lets you change how transparent the color is without having to pick a new color:

```
col.setAlpha((gui.opacity / 100)· 255);
```

This is a small but important detail: the GUI shows opacity as a percentage people understand, but p5 uses numbers from 0 to 255. Changing between these ensures the interface is easy to read and the drawing works correctly.
Finally, the system branches by gui.colorMode:

```
Filled: fill(col); noStroke();
Outline: noFill(); stroke(col);
```

4.2.3 GUI Model, GUI Setup, and Parameter Binding

```
User
 ↓
GUI controls
(Tweakpane)
  ↓
GUI Model (Gui())
(adjustable
parameters only)
  ↓
Shape logic
(radial / polygo-
nal branch)
  ↓
Rendering
fill() / stroke()
  ↓
SVG output
```

The GUI constructor lists all the settings you can change in the generator. It is the one place where all adjustable options are kept. Each property corresponds directly to a visual outcome (e.g., barWidth, cornerRadius), a structural choice (e.g., `shapeType, polygon`), or a rendering mode (e.g., `colorMode, opacity`). Importantly, this object does not include any drawing instructions. It only lists what can be changed, not how things are drawn. This lets the controls, the shape-making code, and the drawing code change separately.

```
function guiSetup() {
gui = new Gui();
gui.generalFolder = new Pane();
```

The `guiSetup()` function links the settings listed in the `Gui()` model to real controls on the screen using Tweakpane, a simple JavaScript tool for changing settings in real time. Tweakpane lets you display things like number ranges, on/off switches, and options interactively. Each setting in the `Gui()` object is connected to a matching control (such as a slider, dropdown, or switch) with clear limits (e.g., minimum and maximum values or predefined choices). This makes sure users can only pick valid options. By using Tweakpane, the system keeps the list of settings separate from how they look on the screen. The generator's main code does not depend on how the controls look; instead, the interface acts as a go-between for the user and the generator. When you change something in the GUI, it updates the shared settings and redraws the design, so you can try out ideas without changing the code directly.

Each `addBinding()` call connects a property in the GUI object to a specific UI control (dropdown, slider, toggle), with constraints such as min/max values or selectable options.

Folders (such as `Shape Settings, Polygon Settings`, and `Concentric`) group related settings, making the generator easier to understand and use.

```
function Gui() {
this.shapeType
= “Radial”;
 this.shape =
“rectangle”;
 this.category
= “Primary”;
 this.selected-
Color = “Blue
Heritage”;
...
```

The interface does not create shapes on its own; instead, it changes settings that trigger the system to redraw. This means the GUI is a middle layer for testing ideas, not a tool for creating shapes directly. The system's highest-level structural decision is:

```
gui.shapeType ≡≡≡ “Radial”
gui.shapeType ≡≡≡ “Polygonal”
```

This one setting determines which shape type is used, but the rest of the process stays the same: both types form a group of shapes that fit within the same border and use the same color rules.

4.2.4 Function draw ()

```
function
draw() {  //
Code to run
repeatedly.
}
```

The generator's visual output is created inside the main `draw()` function. In p5.js, `draw()` is usually defined to run repeatedly to keep the screen updating; the loop is stopped on purpose with `noLoop()` during setup. It's used to create animations and respond to user inputs.

In this system, the loop is intentionally stopped with `noLoop()` during setup. This way, the drawing only changes when something happens, not constantly. So, `draw()` runs once when the sketch starts, and then only when settings change and `redraw()` is called from the control panel. This setup makes the results easier to expect and uses less computer power, while still letting users interact and try things out.

Each redraw begins by clearing the canvas to ensure outputs remain single, clean compositions suitable for export. The first instruction inside `draw()` is:

```
canvas.clear();
translate(width / 2, height / 2);
```

Because the canvas uses the SVG renderer `(createCanvas( ... , SVG))`, `clear()` is used instead of background() to reset the frame every time new shape appears.

```
draw()
|
|
|—radial branch
|   single ring
|   concentric
|   loop
|
|
|—polygonal branch
    single shape
    concentric
    loop
```

This makes sure shapes do not pile up with each redraw and keeps every export as a clean, single image. Then, `translate` is introduced because shapes are built around the origin (0,0) at the center of the canvas. This makes things much simpler: shapes like circles and polygons are easiest to make around a center point, so moving the origin to the middle lets the generator use simple rotation and angle logic without always shifting the coordinates.

Then a defining constant was created for the outer radius:

```
const OUTER_RADIUS = 300;
```

Instead of letting the system grow with the canvas size, a fixed outer radius creates a consistent area that every design must fit into (Fig 4.1). This makes it easy to compare different outputs (which is important when the same system is used for rating, exporting, or creating a dataset), and it also aligns with the idea in identity systems that things should behave the same way across different formats.

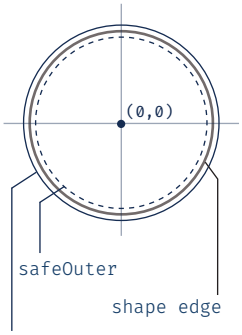


Fig. 4.1 Centered coordinate system and safe drawing boundary.

However, because the system supports both filled and outline drawing, the usable radius varies slightly with stroke thickness. This is why the code calculates:

```
const safeOuter = getSafeOuterForMode();
```

`getSafeOuterForMode()` removes a safety margin based on the outline's thickness (when the system is in outline mode). In outline mode, a thick line can stick out past the edge of a shape. The safety margin is calculated by subtracting half the line thickness, plus a little extra. This ensures that, even with tick outlines, the drawing stays within the intended circle. The line:

```
if (gui.debug) drawDebugUnitCircle()
```

adds an optional guide circle (using a very light stroke color set by `GUIDE_COLOR`) that helps to adjust the shapes step by step, and makes it possible to see the real edge, making mistakes easy to spot, like parts going past the outer circle, things not lining up, or rotation points being off.

4.2.5 Radial mode in the draw function

```
drawRadialEle-
ments(0, 0,
degrees(angle),
ringElements,
w, gui.barWidth2,
h,radius,gui.
shape,
gui.elementRota-
tion);
```

Radial modes arrange elements in a circle by repeating and rotating them. If the concentric option is off, the system creates a single ring of elements within the outer boundary (`safeOuter`). Important settings are how many times elements repeat (count), their size (`barWidth, barHeight`), and their shape (`rectangle, circle, triangle, or trapezoid`). Shapes have parameters of `rotation, starting angle` and outer boundary `(outerRadius)` These settings shape how clear and dense the design looks (Fig 4.2). The angular spacing between elements is determined by:

```
angleStep = TWO_PI / count
```

which ensures equal distribution around the circle. For each iteration, the element's position is calculated using cosine and sine functions, which convert polar coordinates to Cartesian ones. As a result, all elements stay aligned within the same circular layout.

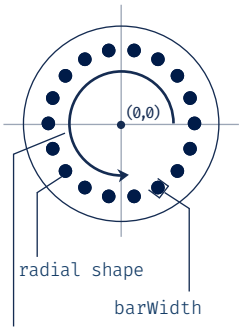


Fig. 4.2 Radial shape and barWidth logic.

Before anything is drawn, the system calculates a bounding box based on the shape. This keeps each element inside the outer edge, so nothing spills out, even if it's rotated or has rounded corners. The system also limits corner rounding to prevent shapes that wouldn't work geometrically.

Instead of setting exact positions by hand, the generator follows geometric rules. This makes the radial system both predictable and flexible. Designers can adjust parameters, and the system maintains consistency.

When concentric mode is enabled, the system can create several rings inside the same outer boundary. Each ring gets a normalized index (`t ranging from 0 to 1`), which defines its relative position between outer edge and center. and then used to set its actual radius using:

```
radius = safeOuter·(1 - t)
```

This mapping ensures that the outermost ring remains close to the boundary while inner rings progressively move toward the

```
strokeW =
lerp(gui.bar-
WidthRange.
min, gui.bar-
WidthRange.
max, tMapped)
```

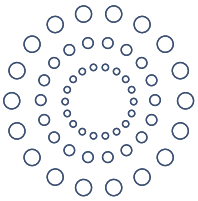


Fig. 4.3 Concetric inward logic.

center. Element dimensions are interpolated between defined minimum and maximum ranges using linear interpolation (`lerp()`):

```
width = lerp(minWidth, maxWidth, tMapped)
height = lerp(minHeight, maxHeight, tMapped)
```

where tMapped controls whether elements grow inward, outward, or remain fixed (Fig. 4.3).
This approach allows multiple structural variations without altering the core generative logic. Additionally, cumulative rotation is applied using:

```
angle = radians(angleOffset * cumulativeElementCount)
```

This introduces gradual rotation, producing spiral alignments that add visual movement while maintaining structural constraints. This is important because it makes the rings feel like they move and connect, rather than just being stacked identically.

4.2.6 Polygonal mode in the draw function

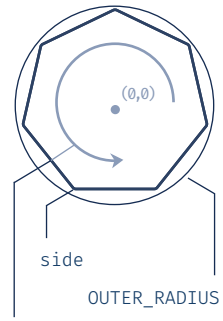


Fig. 4.4 Polygonal shape in the initial position.

Polygonal mode uses the same structure as before but swaps radial elements for geometric shapes such as polygons, stars, or gears. The function `drawPolygonalShape()` defines these shapes through parameters including: number of sides (`sides`), `inner` and `outer radii`, polygon type (`polygon`, `gear`, `star`) and `stroke` thickness and rounding (Fig. 4.4).

Unlike radial mode, polygonal shapes are constructed from connected vertices. For star and gear types, alternating inner and outer points create more complex silhouettes. Because these shapes naturally rotate differently from simple polygons, a base rotation correction is applied:

```
rotationFix = -PI / sides
```

Aligning the top point vertically maintains a consistent look.

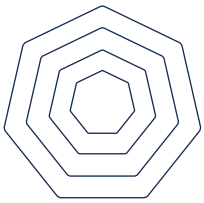


Fig. 4.4 Concetric polygonal repetiton.

When concentric mode is on, polygon shapes repeat inward using the same normalized index system. Because stroke thickness can affect how large these shapes appear, polygon structures require tighter boundary control. So, each ring recalculates a safe outer radius:

```
localSafeOuter = OUTER_RADIUS - getSafetyMargin(strokeWidth)
```

Unlike radial mode, which uses cumulative rotation to create spiral motion, polygonal concentric structures maintain more geometric stability through controlled per-ring rotation (Fig 4.4). Concentric logic acts as a separate layer from geometric construction. This approach lets both radial and polygonal systems use the same structural framework.

4.2.7 Radial Shape Construction

```
function drawRadialElements(cx, cy, startAngle, count, w, w2, h, outerRadius, shape, rotation) {
startAngle = radians(startAngle);
 rotation = radians(rotation);
```

In radial mode, the system generates designs by repeating a single shape around a circle. The main function, `drawRadialElements`, takes a shape (such as a rectangle, circle, triangle, or trapezoid) with set sizes and places copies of it evenly around the circle. This function solves two related problems; how to place a set number of shapes with equal space between them all the way around a circle; and how to place each shape so that its farthest edge stays inside the allowed distance from the center (the safe circle). The function `drawRadialElements` includes:

`(cx, cy)` the center of the entire system (in practice, always 0,0 after translating the origin).

`startAngle` an additional global offset (used especially in concentric mode to create spiral drift). It allows the ring to rotate as a whole without changing distribution.

`Count` is the number of repetitions (elements on the ring).

The element's dimensions: a primary or diameter for circles `(w)`, the top width of a trapezoid `(w2)`, and the element height or radial length for bars `(h)`.

`outerRadius` is the boundary for that ring ("safe" stroke).

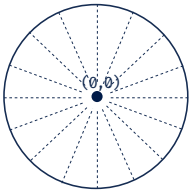


Fig. 4.5 Angular distribution (angleStep).

`Shape` selects which primitive drawing function to use. `Rotation` allows the element itself to rotate around its own center. Elements are evenly distributed by discretizing the full circle into equal angular steps:

```
angleStep= 2πcount
```

Each element is placed at an angle computed from its index and an optional global offset `(startAngle).` This enables controlled rotations of the entire composition without modifying individual element logic (Fig 4.5).

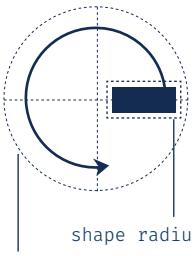


Fig. 4.6 Radial containment.

A key rule in the radial system is that all elements must stay within the circular boundary `(outerRadius)`. Since elements have size and can rotate, placement can't rely only on the radius. The system estimates the space each element takes up using a simple box shape based on the chosen shape. It then moves each element inward just enough to keep it inside the edge. This keeps even rotated or uneven shapes fully inside the circle (Fig. 4.6)

Once the effective placement radius is determined, element positions are calculated using polar coordinates:

```
x = cx + ()r,  y = cy + sin()r
```

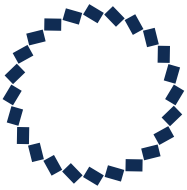


Fig. 4.6 The final result obtained by rotating the elements around their axis and rotating the entire shape

The final direction of each element combines its position around the circle with an extra rotation setting, so the whole design can be rotated without messing up the even spacing.

This structure keeps the layout balanced while still allowing changes to things like size, rotation, and shape. By keeping placement and drawing steps separate, radial mode offers a steady yet flexible setup which allows for further experimentation without losing visual consistency (Fig 4.6).

4.2.8 Polygonal Shape Construction

```
function drawPo-
lygonalShape(cx =
0, cy = 0, outerR
= OUTER_RADIUS,
sides = gui.
sides, polygon-
Type = gui.poly-
gon, strokeW = 1,
t = 0) {
const baseInnerR
= gui.innerRa-
dius;
const innerR =
baseInnerR * (1
- t);
```

Unlike the radial system, where a basic shape is copied around a circle, the polygonal system draws one unbroken outline. In polygonal mode, the generator creates closed shapes using corner points arranged at angles, rather than repeating parts around a circle. This outline's shape depends on how many sides it has and how the inner and outer sizes relate to each other. The polygonal logic supports three closely related shape families:
`Polygon` (regular n-gon),
`Star` (alternating outer and inner vertices),
`Gear` (toothed contour with flat plateaus).

All three use the same basic shape rules and only differ in how the points and angles are arranged. Polygonal shapes are constructed in coordinates around a common center:
`(cx, cy)` – center of the shape.
`outerR` – defining the maximum extent of the shape.
`sides (n)` – number of primary divisions of the circle.
`innerR` – inner radius, used for star and gear shapes.

A full circle is split into equal angles to set where the polygon's corners go (Fig. 4.7). These corners are the points that make up the shape's outline. For a regular polygon, every corner is the same distance from the center (`outerR`). The angle between each corner is:

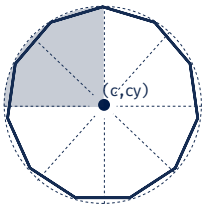


Fig. 4.7 Polygon construction ($\Delta\theta = 2\pi/n$)

$$\Delta\theta = \frac{2\pi}{n}$$

where n is the number of polygon sides. The position of the i-th vertex is computed using polar-to-Cartesian conversion:

$$x_i = cx + \cos(i \cdot \Delta\theta) \cdot outerR$$
$$y_i = cy + \sin(i \cdot \Delta\theta) \cdot outerR$$

After finding these points, they are joined to form a closed shape. Since all points are the same distance from the center and only the angles change, the shape is always regular and looks the same

when you turn it. When building a star polygon, the vertices switch back and forth between an outer radius and an inner radius. This pattern creates the star's pointed look (Fig. 2.6). To achieve this, divide the circle into 2n equal angles, where n is the number of star points. The angle between each vertex is:

$$\Delta\theta_{star} = \frac{2\pi}{2n}$$

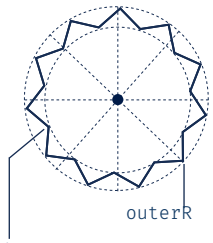


Fig. 4.8 Star shape construction.

Each vertex, labeled by i, sits at one of these angles. Whether it uses the inner or outer radius depends on if i is even or odd:

Even values of i put the vertex on the outer radius (`outerR`), making the star's tips. Odd values of i place the vertex on the inner radius (`innerR`), creating the dips between the tips (Fig 4.8).

By alternating the radius in this way, the system enforces a rhythmic expansion and contraction of the contour, ensuring that every outer point is followed by an inner one. The visual sharpness of the star is therefore controlled directly by the ratio between `innerR` and `outerR`: smaller inner radii produce sharper, deeper spikes, while larger inner radii create flatter, more polygon-like stars.

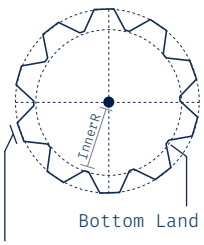


Fig. 4.9 Gear shape construction.

After choosing the radius for each vertex, the same angular subdivision is used, but the radius alternates between `innerR` and `outerR`. This setup ensures all vertices are evenly spaced around the circle, with their distances from the center varying in a regular pattern. The star shape comes from just changing the radius, not from any special drawing rules. The gear shape has flat parts instead of sharp points. Each tooth has: an inner flat segment (`bottom land`), two sloped transitions, an outer flat segment (`top land`) (Fig. 4.9). Each tooth occupies one equal angular division of the circle; `topLand` fraction of the tooth angle assigned to the outer part and `bottomLand` assigned to the inner, define how much of this angle is occupied by flat segments. The remaining angle is divided equally between the two sloped parts. By putting the flat parts in the middle of each tooth and gap, the gear looks balanced (Fig. 4.10). To make the outline, alternate between points on `innerR` at the ends of the bottom part and points on `outerR` at the start and

Fig. 4.10 The final result obtained by combining the parameters Top Land, Bottom Land, InnerR in gear mode.

end of top part. This process creates an outline that looks like a machine part and is always built the same way using math. The inner radius is constrained to remain sufficiently smaller than the outer radius to prevent self-intersection. This ensures that inner vertices never cross outer edges. This keeps the shape drawable for any valid set of parameters. After building the shape, rotate it to make sure it lines up visually: regular polygons are aligned vertically, stars and gears are rotated by - n.
This change ensures a point or tooth stands straight up rather than sideways, making the shapes easier to see and more consistent.

4.2.9 Rounded Corner Logic

```
function draw-
RoundedPoly-
gon(points,r)
{
const n =
points.length;
beginShape();
```

Rounded corners are added in a separate step that changes a shape's outline but keeps its basic structure. This method replaces sharp corners with curves, so the shape looks the same overall but has smoother edges.

This approach allows the same rounding process to be applied to many shapes, such as rectangles, triangles, polygons, stars, and gears, while still respecting each shape's rules. For every corner, the system looks at three points in a row: the previous vertex (Vp), the current vertex (Vc), and the next vertex (Vn). These three points form an angle (β) at the current corner. Rounding turns this angle into a smooth curve that connects both nearby edges (Fig 4.11). The angle's geometry sets the details. Given the angle at the corner, the distance from the corner to the tangent points is:

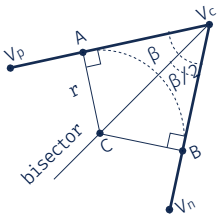


Figure 4.11 Geometric construction of a rounded corner using three consecutive vertices.

```
d=r tan(θ/2)
```

The radius is automatically limited so it never exceeds available edge length. Once the right points are found, the curve's center is set along the line that bisects the angle. The arc is constructed manually to ensure consistent SVG export and rendering precision. Instead of using built-in curve tools, the system builds the arc directly by interpolating between points. This gives full control over the level of detail and avoids differences between rendering

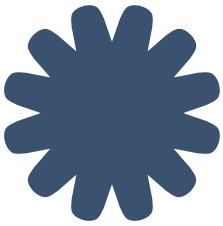


Fig. 4.12 Rounded corners applied to polygonal gear structure.

Fig. 4.13 Rounded corners applied to radial rectangular elements.

systems. Not all vertices are eligible for rounding. To preserve structural clarity:

Polygons allow rounding on all vertices.

Stars allow rounding only on outer vertices.

Gears allow rounding only on outer tooth tips.

This rule prevents unwanted shapes, such as rounded dips between star points or inside gear gaps, that would make the shape look wrong. Formally, rounding is conditionally applied based on a vertex tag. Each `vertex` is tagged as eligible or ineligible for rounding depending on shape type. This tagging system keeps the way shapes are built separate from how they are styled, so rounding works in a clear and controlled way (Fig 4.12- 13). The size of the rounded corners is automatically limited to prevent breaking. For star shapes, an additional constraint links the rounding radius to the difference between outer and inner radii:

```
rmax≤0.45(outerR-innerR)
```

This constraint prevents thin spikes from collapsing when corner rounding becomes too large Rounded corners are added at the end to change its appearance without altering the main shape. By adding rounding after creating the shape, we keep the shape and the style separate, making the system easier to handle.

4.2.10 Helper Functions and Supporting Logic

In addition to the main drawing functions, the shape generator uses helper functions to keep the system running smoothly and consistently in different situations. These helpers do not add new visual features. Instead, they handle constraints, keep things organized, and do background calculations that would otherwise clutter the main code.

These helpers make the system easier to understand, update, and adapt, especially as more settings are added and start to interact in more complex ways. At the system level, fixed constants define shared constraints:

GUI parameter
gui.strokeWight
colorMode
↓
Helper function
getSafetyMargin()
getSafeOuterFor-
Mode()
calculateBound-
ingBox()
↓
Safe values
↓
Drawing function
drawRadialEle-
ments()
drawPolygo-
nalShape()

```
const OUTER_RADIUS = 300;
const GUIDE_COLOR = “rgba(0, 0, 0, 0.1)”;
```

OUTER_RADIUS sets a common starting point for all shapes. Instead of recalculating the edges every time, this value serves as a stable base for all layout decisions. This makes it easier to think about space and ensures that changes in one area do not accidentally affect other parts.

GUIDE_COLOR is used exclusively for debug visualization. Keeping debug-related values separate from production parameters avoids accidental coupling between visual output and development tools.

Several helper functions ensure shapes remain within allowed limits. Instead of putting safety checks in every drawing function, the system puts this logic in one place:

```
function getSafetyMargin(strokeW = 0) {
return (strokeW / 2) + 2; }
```

This helper turns a visual setting (stroke weight) into a space limit. By putting this calculation into a single function, the system avoids code duplication and keeps the edge rules consistent across different drawing styles. Another helper works out the actual drawing size based on the current settings:

```
getSafeOuterForMode() {
const s = (gui.colorMode=== “Outline”) ?gui.strokeWeight: 0;
return OUTER_RADIUS - getSafetyMargin(s);}
```

Here, visual settings directly influence geometric constraints, demonstrating how interface parameters are converted into structured layout rules using small, specific helper functions.

In circular layouts, where things are placed around a center, placement depends on where something is, how big it is, and how it is turned. To handle this, the system uses a helper that guesses how much space a shape takes up after it is moved or rotated:

```
calculateTrapezoidBoundingBox( ... )
```

Instead of doing exact math to check whether shapes overlap, this function provides a useful estimate based on the shape's box after it is rotated. The result is not used for drawing but for fixing placement, so shapes can be moved inward if needed. These choices are intentional and fit the system's role as a tool for testing designs, not for perfect geometry. exploration tool rather than a geometric engine. The helpers ensure that stored configurations are reproducible and random changes do not mess up the system's settings. By isolating these concerns, the system keeps the generation logic independent of the storage logic, even though they coexist in the same codebase. The system includes optional debug helpers such as:

```
drawDebugUnitCircle()
drawDebugBoundingBox()
drawDebugCentroid()
```

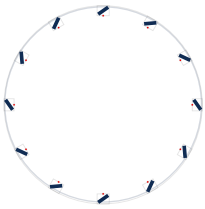


Fig. 4.14 Debug mode visualization showing structural boundaries and placement guides.

When debug mode is activated, auxiliary visual elements appear to reveal the internal logic of the generator (Fig 4.14). These include bounding boxes, reference centroids, and safety boundaries used to ensure elements remain within the allowed radius. The overlay is used only during development and does not affect the final visual outpu. Helpers are extra tools that make design choice easier.

4.2.11 Output Generation and Export

In this project, making the final image happens as part of building the shape. The same code that arranges the shapes and applies the rules also draws the final image.
The generator works inside the `draw() function`, which is the main part that draws everything. Before the drawing function starts, the program's starting point is the center of the canvas. This way, all shapes are built around the same center, making it easier to handle both circles and polygons. Because of this, shape formulas can use `(0, 0)` as the center, which means fewer position changes and a lower risk of numerical errors.

```
GUI parameters
(fill / out-
line / opacity/
stroke)
    ↓
Rendering setup
(centered coor-
dinate system)
(getSafeOuter-
ForMode())
    ↓
Shape generation
(radial / polyg-
onal)
    ↓
SVG renderer
(createCanvas
SVG)
    ↓
ExportSVG()
(final vector
output)
```

A few main settings from the current parameters control how each output looks.The settings from the current parameters that control each output are fill or outline mode, stroke weight and color with opacity. These settings are set once at the start of each drawing cycle, before making the shapes. This keeps the look consistent and avoids repeating code changes. Color and transparency are handled by turning the chosen color into an RGBA value and changing the alpha channel like this:

```
col.setAlpha((gui.opacity / 100) * 255);
```

By keeping color choice and transparency separate, the system lets you use the same colors for different levels of transparency without having to redefine them. A key part of producing the output is ensuring no shapes go outside the area they are supposed to stay in. This is done by using a safe outer edge, which is worked out as the shapes are being drawn:

```
eOuter = getSafeOuterForMode();
```

When outlines are turned on, the line thickness makes each shape take up more space. The system handles this by making the allowed area a little smaller. This ensures the final images always stay within the areas they are supposed to, no matter how thick the lines are or how the shapes are rotated. This rule applies the same way to all elements, shapes with many sides, and shapes with the same center. All exported images fit inside the same circle and can be safely used in different formats and situations.

The canvas starts in SVG mode, so images don't lose quality when you scale them up or down. All shapes are vector images, so you can resize them without losing their appearance. Details like lines, points, and curves stay the same in the exported file, making the results easy to edit, use in designs, or use for branding. The feature is kept simple on purpose:

```
function exportSVG() {
save("radial_elements.svg"); }
```

Export does not reapply the shapes or make any additional changes. Instead, it saves exactly what the generator has already drawn. This supports the idea that the output is not a separate thing, but a direct result of the rules and settings used. In this system, each exported image shows one version of the rules with a certain set of settings. There is no difference between a preview and a final version: what you see on the screen is already ready to use. This way of working aligns with the generator's main goal as a design tool, not just as a way to show ideas. The outputs are not just rough drawings but organized graphic objects that can be used directly in layouts, branding, or other computer processes.

System Summary

Technically, every part of this section explained how the shape generator works as a system that follows rules to build visual shapes. By using methods to create circles and polygons and additional helper tools, the generator creates a space where different shapes can be made with various parameters. Instead of producing a single result, the system sets limits, connections, and rules for changing shapes that determine what can be made. Random changes and different settings are controlled to keep everything clear and organized. Although the system structure defines deterministic generative rules, parameter randomization introduces an exploratory entry point, discussed later within the interaction layer. Helper tools support this by handling where things go, ensuring sizes match, and fixing problems, so the shapes made always look right and are easy to understand across many different setups. Parameter updates trigger `redraw()`, ensuring that every change in the interface immediately regenerates the output. These features are separate from how a designer actually uses and changes them. The question of how this system is useful in real life and which parts are shown, hidden, or highlighted remains unanswered. This shift from internal structure to interaction is described in the next section.

4.3 Interface and User Controls

"A user interface is not a presentation of the internal structure of a system; it is a means of enabling people to think and act using that system." (Bret, 2010)

The generative logic and architectural structure form a complete internal system, however, technical completeness does not automatically translate into meaningful interaction.

How the rules and settings are connected within the system does not determine how the designer experiences, understands, or interacts with the system in a meaningful way. This is where interface logic comes in.

The focus moves from how the system is built to how it is used. Describing the coding part behind the parameters shows what a generator can produce, while the user control examines how these options are discovered, managed, and experienced through the interface. The interface shows how the system works internally and helps organize it, giving the designer a chance to explore and make choices within it.

With generative design tools, complexity is carefully managed to help users explore rather than make things too confusing. The shape generator's interface acts as a control layer, allowing designers to explore and make decisions without needing to understand the system's internal workings. Instead of revealing complex details, it offers key parameters so designers can control the generative process without coding.

The interface uses Tweakpane, a lightweight GUI library designed for real-time parameter manipulation. Tweakpane has simple controls, including sliders, drop-down menus, switches, and grouped folders, that are easy for designers and programmers to manage. The system is designed to reduce the cognitive effort required to learn how to use the tools. So, no new pattern was invented; the one most convenient for this type of research was used.

Controls are organized hierarchically and conditionally. Parameters appear or disappear depending on the selection, making it easier for the designer to work in different settings, such as radial or polygonal forms, because these settings are different and never used together. The parameters, therefore, have different priorities and actively shape the way the design space is navigated. Hierarchically and well-organized interfaces often work better

than complex ones, especially when designers need to review or explore options multiple times. Clear controls, predictable actions, and easy use help designers stay focused and avoid mistakes or confusion (Munro, 2021).

It is important to mention that this is not the final appearance of the interface but just part of the design process and research. It is set up so that all settings can be seen, and to make visible how choices in the interface affect usability of the system and the quality of the design decisions.

4.3.1 Parameter Visibility and Interaction Logic

The interface does not show every internal setting; it offers a selected group of options chosen to match design goals, not just to show everything the code can do. Parameters are treated as interaction elements rather than technical settings.

In the code, settings are defined within `Gui()`, which keeps track of both the code that creates shapes and the interface. The settings that are connected to interface controls using the Tweakpane library are not appearing all at the same time automatically; only settings that matter for design choices are shown, while others stay hidden or are set by the code itself.

For example, the `OUTER_RADIUS` setting sets the maximum size for all shapes generated by the generator. This value is intentionally set and is never displayed in the interface. Letting users change it would make it hard to compare results and could cause problems with size. Instead, designers work within a fixed space, while built-in functions (such as extra space for line thickness) ensure the shapes look right.

On the other hand, settings like elements, `barWidth`, and `barHeight` are shown because they align with easy-to-understand design principles: repeating parts, thickness, and how far shapes extend from the center. Designers can adjust these settings with sliders to try out different looks while staying within the system's safe limits. Each setting shown has a range, step size, and limit. These limits are not random but are based on what was learned

while building and testing the system. For example, the setting that controls the number of radial elements is limited to between 4 and 200. If it falls below 4, the shapes become unclear; if it rises above 200, the shapes create ambiguous forms, and the parts blend together (Fig. 4.15-16)By setting this range in the interface, the system prevents users from creating shapes that do not work, while still letting them try many options.

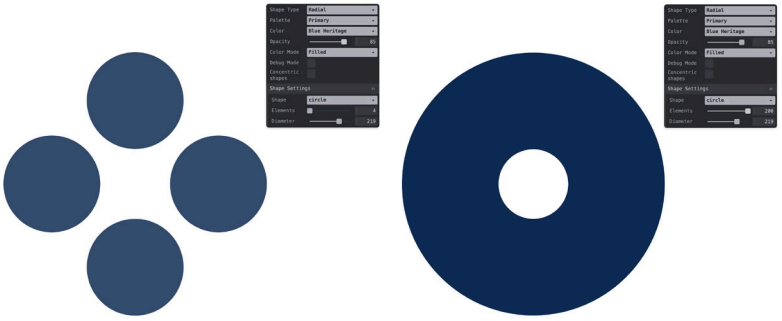


Fig. 4.15–16 Extreme parameter constraints for radial element count. Minimum (left) and maximum (right) allowed values demonstrate how interface limits prevent visually ambiguous or structurally unclear outcomes. element (circle).

Similarly, corner rounding parameters (`cornerRadius` and `polygon-specific rounding values`) are capped based on geometric feasibility. In the case of radial rectangles and triangles, the corner radius is dynamically constrained to prevent overlaps or self-intersections when shapes are rotated. In polygonal modes, rounding is further restricted based on whether a vertex represents an outer or inner point, ensuring that only visually coherent edges are softened. These limits mean users do not have to guess what values will work. Instead, the interface embeds design knowledge into its operation, helping users create shapes that are always correct. Shown settings change depending on the system mode. The generator works with two main shape types, radial and polygonal, and the interface changes to match the current type. When the system is set to radial mode, settings for repeating parts (`elements`), bar size (`barWidth, barHeight`), and shape choice (`rectangle, circle, triangle, trapezoid`) are shown. Controls for just

polygons, such as the number of sides or inner radius, are hidden. When polygonal mode is chosen, the controls for radial shapes are removed and replaced with settings such as sides, `innerRadius, topLand, and bottomLand`. Instead of showing all controls at once, the interface changes to fit what the user is working on. The interface does not require designers to understand the full system upfront; instead, it reveals complexity incrementally as choices are made. Reducing cognitive load by aligning visible controls with the user's current task. The interface does not require designers to understand the full system upfront; instead, it reveals complexity incrementally as choices are made (Sharp et al., 2019). The interface logic is shaped by how parameters turn on or off, or change one another, as you interact. These changes are not just visual, they also reflect how the system generates shapes.

```
START
  ↓
Mode selection
  ↓
Radial controls
appear
  ↓
Advanced options
appear
(rounding / con-
centric)
```

4.3.2 Radial Mode

When the user selects Radial as the shape type, the system activates a generative logic based on angular repetition around a central point. At the interface level, this immediately exposes `elements` (number of repetitions), `barWidth` and `barHeight` (radial thickness and length), and the selection of shapes (`rectangle, circle, triangle, trapezoid`).

At the same time, all polygon-specific settings, such as sides, `innerRadius, topLand, bottomLand`, are hidden. This keeps things clear: radial shapes are made by repeating parts around a center, not by connecting points. If the user changes the basic shape in radial mode, for example, from circle to triangle, the interface updates again to show or hide the right settings. Within the radial mode, rotation and corner-rounding controls are visible for triangles, trapezoids, and rectangles, but remain disabled for circles, where these settings do not matter. This way, users only see controls that actually affect the current shape.

For rectangles, trapezoids, and triangles, turning on `roundedCorners` displays additional controls that let users choose which corners to round. These options are hidden for circles, where

rounding obviously is not needed. These options are not just for appearance; they are based on what shapes are possible. For example, the corner radius is always kept within safe limits for the shape's size, so rounding never goes beyond what the shape can handle when rotated and placed in the pattern (Fig. 4.17-18).

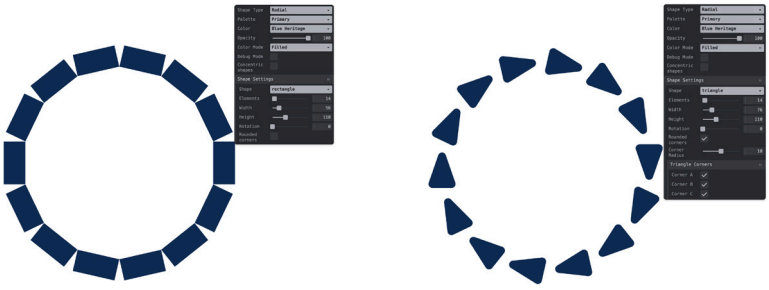


Fig. 4.17—18 Conditional parameter visibility within radial mode.
Interface controls adapt to the selected shape. Only radial controls are displayed, while polygonal parameters remain hidden.

4.3.3 Polygonal Mode

When the user switches to `Polygonal mode`, the way you work with the system changes. Instead of repeating separate parts, the system now produces a single continuous outline using points.

As a result, radial parameters are replaced by controls such as sides, which define the number of vertices; `innerRadius`, which determines the indentation depth for stars and gears; and `topLand/bottomLand`, which modulate tooth and gap widths in gear shapes. Picking different polygon types also changes which controls you see. A regular polygon shows only the side setting, since all points lie on the same circle. Choosing a star adds `innerRadius`, which lets you set the depth of the points. Picking a gear shape adds topLand and bottomLand, allowing you to adjust the width of the teeth and gaps. These changes show how the system shifts from adjusting size and repetition in radial mode to changing the outline in polygonal mode. The interface makes this clear by only showing settings that matter for the current shape (Fig. 4. 19-20).

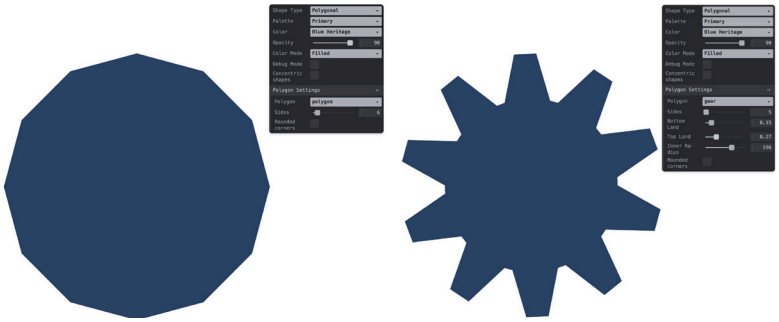


Fig. 4.19—20 Conditional parameter visibility depending on polygonal shape type.

4.3.4 Concentric Mode

The concentric option works as an extra layer, not a separate mode. When turned on, it repeats the same shape process across multiple layers of different sizes for both radial and polygonal shapes. Enabling this option displays settings such as rings, `angleOffset`, and size ranges. These settings control how the main shape is copied in or out from the main edge. Because polygonal shapes are continuous outlines, concentric compositions become visible mainly when stroke mode is enabled. This is intentionally left as a manual choice, allowing layering effects to also emerge through opacity adjustments.

The original fixed width and height values were replaced with width and height range sliders, enabling size interpolation across concentric layers instead of static scaling. The size behavior across concentric layers is managed through a `size mode` selector, allowing users to choose between `fixed`, outward (`center to edge`), or inward (`edge to center`) scaling. This translates the previously defined interpolation logic into a clear directional control within the interface. In polygonal shapes, a `stroke range` parameter is also introduced, allowing stroke thickness to interpolate across layers.

When the concentric mode is off, all these controls are hidden, and the system reverts to a single layer without changing the main shape. Simple layering makes it easy to pick the shape first, then to decide its basic appearance, and last, how many times it is repeated optionally creating concentric form (Fig. 4.21-22).

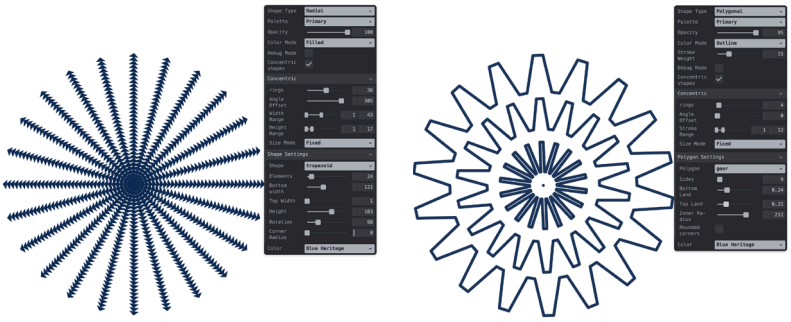


Fig. 4.21—22 Concentric mode applied to radial and polygonal shapes.

4.3.5 Interaction Logic and Interface Feedback

```
User input
(slider /
dropdown change)
    ↓
GUI state
updade
(shared Gui()
model)
    ↓
Redraw trigger
(redraw())
    ↓
Draw function
(drawRadial /
drawPolygonal)
    ↓
Visual output
update
(new shape
displayed)
```

Each interaction directly updates the shared state model and produces a new visual outcome. In all modes, each parameter change triggers an immediate redraw in the code, not necessarily changing the shape, but adding what is dictated by the newly added parameter. If a polygonal shape is included with the radial, the shape is redrawn and updated completely, returning to the polygon's basic neutral parameters.

The system never produces configurations that conflict with the interface's visible state. Designers do not have to commit or make changes; they just keep tweaking the settings and seeing what happens.

The layers of the interface are built for exploration in the research part. The system does not try to find a 'best' solution or judge the results at this stage. The interface just ensures that each change produces a valid, clear visual result. It encourages users to compare, experiment, and try variations rather than edit the same idea.

Although the shape generator does not yet adapt to learning technology to give more reliable results, its interface already follows these principles of responsiveness and clear system state by making every change visible and understandable (Amershi et al., 2019).

4.3.6 Randomization of Parameters

The randomization is implemented as a dedicated button within the GUI system, positioned separately from the adjustable parameters and directly below the generated object. Each click produces a completely new combination of parameters and immediately redraws the output. This function lets users experiment by randomly varying all parameters. While this temporarily removes precise user control, designers can immediately refine the generated result through manual parameter adjustments.

Each activation generates a new configuration that may either align with the branding logic or visibly diverge from it. This leads to problems where the combined parameters do not achieve the visual appearance in accordance with the brand, and sometimes even result in "errors" in every sense of the design. This shows an important difference: the generator can make visual results without the user doing anything, but these results are not always useful for branding design. Now, randomization is more for exploring what the system can make, not for following user choices or rules.

This independence from random settings raises questions about the current parameters, their combinations, and how the program itself cannot assemble the visual appearance of an object that a designer can choose from, given the branding guidance (Fig. 4.23). The program's freedom results in outcomes that deviate from the parameters defined in the branding manual. This contrast highlights the limits of uncontrolled randomness and motivates the later evaluation phase of the research.

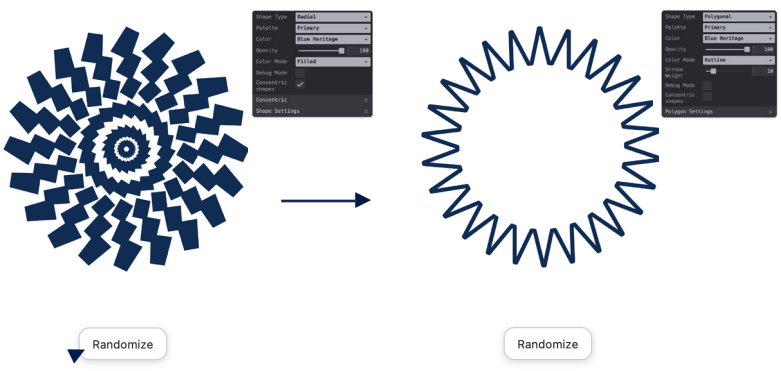


Fig 4.23 Randomization interaction. The randomize button generates a new parameter configuration independent of the current state, producing a new visual output that may align with or diverge

4.4 Limitations of a Rule-Based Generator

Even though the shape generator is flexible and works well inside its own rules, it is still a rule-based system. This is both its strength and its weakness. Rules help keep things consistent, controlled, and repeatable, but they also set limits that cannot be changed unless you add outside judgment or ways for the system to adapt.

The limitations in this section are not mistakes, but are built into the way the system works. Some are kept on purpose, while others show where the system reaches the limit of what rule-based generation can do on its own.

The generator only uses set parameters and geometric rules. Because of this, every shape it makes follows the rules: shapes stay within set spaces, follow boundaries, and fit the system's logic. However, the system does not understand meaning, context, or what is suitable.

The system sees all generated designs as equally good. A crowded design and a simple one, a balanced shape and an awkward one, are all fine as long as they follow the rules. The system cannot tell which designs fit branding goals and which do not.

This limitation is basic to rule-based systems: rules decide what is allowed, not what is better. Even though the parameters are carefully set and shown in the interface, the system treats them all as equally important. Things like the number of elements, the shape used, rotation, or how corners are rounded are all handled the same way. In reality, not all parameters affect the design equally. Some changes have a big impact on recognition and how well things fit together, while others only make small differences. The generator cannot decide which differences matter more. It cannot tell which settings are more important in a given situation or for a specific user (Saadi & Yang, 2023)

Treating all settings the same is a choice for now, but it also shows a weakness: without more input, the system cannot learn which types of changes matter more than others. Randomness helps the generator explore different designs and find new possibilities. But right now, the random choices do not notice if the results are good or bad.

Randomly made designs are just as likely to be a bad fit as they are to be interesting. The system does not remember past results, does not know what works or what does not, and cannot guide random choices toward better designs. This limitation reinforces a broader issue: generation alone does not equal design. Without feedback, repetition simply accumulates variation rather than knowledge. Maybe the biggest problem is that there is no way for the system to get feedback. After a design is made, the system receives no information about how designers see it, choose it, or reject it.

The generator's behavior does not change over time, and that behavior is normal for rule-based systems and is not a technical error. However, it becomes a problem when the system is used long-term in a brand, where improvements over time and changes are necessary. Taken together, these problems show that while rule-based generation is good at creating clear options, it cannot account for personal taste, judgment, or what fits a specific situation.

Rather than trying to solve these problems by adding more rules, the project treats them as signs that human input is needed. To overcome the limitations of a rule based system, integrating external understanding mechanisms such as user-defined inputs or machine learning models become important in order to introduce higher-level evaluation criteria.

This leads to the next chapter, which introduces structured evaluation to support rule-based generation. By adding human judgment, the generator can begin to distinguish outcomes, identify preference patterns, and shift from random variation to informed adaptation.

PART III

Evaluation & Analyzation & Future

05

Data Collection and Evaluation

The generator described in the previous chapter produces consistent visual results, adheres to geometric rules, and allows designers to explore different options by adjusting parameters. But along with the established limits of rule-based models that cannot be changed at this stage of the generator, there are also those that can be avoided. The questions need to be answered: are the generated results meaningful, usable, or suitable for designers? The system can create forms, but it cannot decide which is better. This limitation is built into rule-based systems. While rules set the range of possible results, they do not include judgment. Choices like how visually consistent something looks, whether it fits a brand, or how good it seems are still personal opinions and cannot be decided just by shapes. To go beyond a fixed set of results, the system needs to work with organized human feedback.Evaluation is not just an extra step; it is how subjective judgments are turned into useful data. Annotation means turning human feedback into training material for future system improvements (Munro, 2021). Even before adding learning features, evaluation connects how the system works with how people see its results.Human feedback depends a lot on how the interface looks, how tasks are set up, and the situation. People are influenced by the order of choices, by repeating the same task, and by how options are presented. Because of this, evaluation cannot just be added to an existing system. Both the generator and its interface must be carefully adjusted to make sure judgments are steady, clear, and trustworthy. This brings a key part of the research phase into focus, evaluation. The following sections explain how the generator is modified to support structured evaluation, how user feedback is collected, and how choices in interaction and evaluation affect data quality. This is where evaluation becomes a key part of future system.

5.1 System Adaptation for Evaluation

So far, the shape generator has been described as a tool for experimenting and making controlled changes. Shapes are created according to set rules and you can adjust settings through a menu. But this setup, by itself, does not allow the system to capture how designers see, judge, or value the shapes it creates. To allow for evaluation, the system needed specific changes that turned it from just a tool that makes shapes into one that can collect feedback from people. The system is adapted to evaluate only random shape generation. This change does not yet add features that help the system learn or get better. Instead, it adds tools to record and sort parameters. The goal here is not to change how shapes are made, but to change how results are looked at, judged, and saved. During research, settings can change frequently, and each new setting replaces the previous one. That is why the system uses simple commands. When the user opens the program, a shape is already there; to see the next one, the user must rate it. The goal was to obtain a large number of useful samples quickly. This design choice ensures that each rating corresponds to a specific, unchanging setup (Fig. 5.1). It follows ideas from systems that include people in the process, making sure that judgment is done on clear, fixed choices to avoid confusion and unfairness.

Fig. 5.1 Changed interface system with added rating scale and rating storage for evaluation

5.1.1 Integrating Rating Controls into the Interface

The evaluation screen adds a special area just for ratings to the current settings menu. The GUI settings menu is completely hidden in this part, so users can focus only on ratings.

Instead of adding more to the existing controls, judging is done with a small, clearly separate set of buttons below the main area with randomly generated shape. Several changes were made to the interface to make the system easier to test. The interface was changed to create a list of results, making it easier to review the data after testing. It was also adapted to be used both on phones and with a keyboard, making rating faster and more consistent. Finally, the shapes were deliberately limited because early tests showed that outline shapes are rarely used in Politecnico di Milano style, which could have made the test results less clear.

The rating system uses a five-point scale, from Very unsuitable to Very suitable. This choice is based on research showing that simple scales with a few options are more reliable for personal opinions than sliders, especially when many people are rating. By limiting answers to a few choices, the system makes things clearer and more consistent for everyone. Importantly, the rating buttons do not change the shape. Their only job is to record what the user thinks. This clear split helps keep design actions (such as changing settings) separate. Importantly, the rating buttons do not change the shape. Their only job is to record what the user thinks. This clear split helps keep changes to the shape and opinions separate. without more information. To ensure the results can be understood later, the system saves not just the rating but also all the settings that created the shape being judged. Each saved rating includes:

```
Shape Displayed
    ↓
User Rating
(1–5)
    ↓
Stored Entry
(rating +
parameters +
mode + time-
stamp)
    ↓
Export Button
    ↓
Structured File
(CSV / JSON)
```

the numerical rating,
the active shape mode (radial or polygonal),
all relevant parameter values,
a timestamp identifying when the evaluation occurred.

This way of doing things follows good practices for collecting feedback, making sure extra information is available to understand what people think and to notice patterns in their answers.

The intention was to later find patterns in high- and low-rated forms, so that they could be sorted into a database and, from that, inferred which settings were frequently repeated. The judging process is kept simple on purpose. At any time, the user sees just one shape and a few rating choices. No extra menus, text boxes, or explanations are needed. This design follows the idea that it is best to keep things easy to understand when people have to make the same kind of judgment repeatedly (Gillies et al., 2016; Munro, 2021). The interface uses familiar interaction patterns, like buttons with clear labels and instant visual feedback, so users can focus on judging the shapes rather than figuring out how to use the system. This simple setup is especially important because judging will happen many times for different shapes.

5.1.2 Exporting Evaluation Data

```
function saveRatingsToLocalStorage() {
try {
localStorage.setItem(RATING_STORAGE_KEY, JSON.stringify(ratingData));
 } catch (e) {
console.error("localStorage write failed:", e);
  }
}
```

The system includes one of its most important export functions, which stores all the ratings and related information in a structured file. The data is exported so that, as someone evaluates, they see the exact number of ratings. When they have reached the relevant number of samples, they click the export button, and everything is saved in a single file for easy analysis later with tools such as spreadsheets, charts, or graphs.

By separating the analysis from the generator, the system maintains a clear distinction between data collection and data understanding. This helps ensure that the generator is viewed as a tool for experimentation, not as a system that makes decisions on its own.With these changes, the system moves from being just a tool for looking at shapes to one that can also support organized judging. The way shapes are made stays the same, but now the results are part of a controlled process that lets the system collect people's opinions in an organized way.

This change sets the stage for what comes next. With ways to judge shapes now built in, the system can be evaluated not just for what it produces, but also for how people respond to those results. This opens the door for analysis, comparison, and thinking.

5.2 Rating Logic and Data Collection

```
Generated Shape
      ↓
Expert Judgment
("How suitable
for Polimi
branding?")
      ↓
Scale (1–5)
      ↓
Structured
Rating Record
      ↓
Dataset of
Ratings
      ↓
Pattern Analysis
      ↓
Identification
of Preference
Trends
```

This research turns expert opinions into organized data that can be studied. Rather than treating evaluation as just a last step, the rating process is used as a primary way for experts to guide the generative system's growth.

Here, evaluation connects the system's rules to how designs work in real life. Just following the rules does not guarantee results that fit the brand or situation. In professional design, especially for institutions, suitability is not just a yes-or-no answer; it is more like a range shaped by experience, common practice, and goals.

For this reason, a five-point rating scale was used, ranging from Very unsuitable (1) to Very suitable (5). This allows evaluators to give detailed feedback without being too precise. The importance of Politecnico branding is emphasized in the evaluation process. Instead of requiring evaluators to edit shapes or define new rules, the system poses a single, repeated question: How suitable is this generated shape for use within the Politecnico di Milano visual identity? To simplify the designer's reasoning, the system uses a simple scale. Using scales, rather than sliders or open-ended feedback, helps people stay consistent and makes the process less tiring. This approach matches established data annotation practices, making it easier to organize and compare data later (Wang et al., 2019).

5.2.1 Importance of the Communication Office as Evaluators

As already mentioned, this is a closed system used only within a specific institution. Because of this, the evaluator needs to have experience and regular contact with the institution's design, so only the Design Communication Office of the Politecnico University did the evaluation. Only those who work with the brand every day know how to keep it consistent. The institution's visual identity is kept through professional experience, internal rules, and ongoing design work. Because of this, the Communication Office

staff are the experts, and their opinions carry special importance in the organization. The evaluation does not present a survey of personal preferences. This aligns with Munro's point of view that complex or context-dependent tasks require trained evaluators, not large groups (Gillies et al., 2016).

The evaluation document sent to the Communication Office is important for the process. It explains what the task covers, its limits, and the rules for doing it fairly:

The task is clearly explained: shapes are drawn from the Polimi visual style and judged for a made-up Welcome Week event, so opinions are based on a realistic yet controlled situation. The rating instructions clearly define the scale and provide visual examples of both suitable and unsuitable uses, in line with the official brand manual. Data transparency ensures that no personal data is collected and clarifies that only generative parameters and ratings are stored. This setup clarifies things and reduces the risk that evaluators will use different standards. Research shows that clear instructions make collected opinions more trustworthy.

To make sure the evaluation was consistent and clear, a special document was sent to the Politecnico di Milano Communication Office. This document gave instructions and background, so all evaluators knew what the task was, its goals, and its limits.

Instead of calling the generator a test or technical tool, the document describes it as a design helper seeking new ways to expand the Polimi visual identity. This matters because evaluators are judging how shapes could be used in branding, not just as random designs. To make the assessment as accurate as possible, the document sets out the actual situation for which the test is being conducted, including Welcome Week materials, posters, banners, and visual details used by the university for the occasion. This creates a clear boundary between personal taste in the selection of possible applications. A key part of the document is the explanation of the five-point assessment scale:

```
1 – Very unsuitable  2 – Unsuitable  3 – Neutral  4 – Suitable 5 - Very Suitable
```

Each level is explained in terms of how well it fits the brand, not just its overall quality. For example: very unsuitable shapes are those that do not match the brand's look, are too complex, or are hard to use in different environments. Very suitable shapes are those that can be used right away in communication materials without any changes. This explicit explanation serves two purposes:
It reduces ambiguity in interpretation. It helps ensure people judge shapes by the institution's standards, not just what they personally like. Shapes are rated based on how well they can be used in Politecnico di Milano design situations, such as event announcements, where fitting the context is more important than appearance alone. The document provides clear visual examples of how the generated shapes might appear in branding contexts. These examples help evaluators make their decisions. The examples are shown as reference points, there is no 'correct answers'. Evaluators are encouraged to use their expertise and the examples to maintain consistent ratings (Fig. 5.2). Shapes that follow Polymysian geometry and fit with existing shapes in the application are marked as examples and receive the highest evaluation (5). Shapes that add too many elements, overlapping and looking messy, are shown as negative examples with the lowest evaluation (1).

High suitability (5)

Low suitability (1)

Fig 5.2 Evaluation reference set used to calibrate expert judgment.

This method follows expert best practices, where having shared examples helps evaluators agree more often without needing strict rules. In the document it is clear that evaluators are not expected to design, modify, or suggest improvements. Their role is limited to assessment, not authorship, and the ratings are saved for later study. This separation helps evaluators focus on suitability and avoids extra work. The document explicitly addresses data collection and transparency:
Only ratings and associated generative parameters are stored.
No personal data is collected.
No images are permanently stored; shapes can always be regenerated from parameters.

Being open about what is collected supports ethical evaluation and builds trust in the process. This also ensures that each rating can be matched to the exact settings used, allowing for an organized study rather than just guessing. The examples labeled as very suitable and very unsuitable serve as key reference points for later analysis. The scale captures: marginal cases, ambiguous outputs, shapes that are technically correct but do not fit the context well. This is important for later steps, where ratings are grouped and patterns are carefully studied. Overall, the evaluation document links the generative system to real design work. It turns computer-made results into something design professionals can judge, while keeping enough structure for later study. By carefully defining context, scale, roles, and examples, the document ensures that collected ratings are suitable for transformation into structured datasets. This practical setup leads to the next section, which explains how the collected ratings are organized, stored, and analyzed.

Each evaluation creates a structured record that includes: the full set of parameters used to generate the shape, the selected rating (1–5), a timestamp associated with the evaluation session. The exported rating files directly connect the parameters that are not visible through evaluation. The system is able to recreate visual input based on parameters so no images are saved. This makes it easy to track, repeat, and group the results by shape type, number of lines from the center, shape complexity, and size relationships.

While this system still operates according to set rules, its evaluation process is much like that of systems that can learn and adapt. Expert opinions are turned into data, so the generator can be tested, its results grouped, and the system improved. The rating process does not restrict design choices but helps clarify how they are arranged. The next chapter will look at how the system is changed to support evaluation and how the ratings are organized for analysis and future updates.

Table 5.1 Example of Stored Rating Record

Field	Example Value
Timestamp	1770730710970
Rating	4
Shape Type	Radial
Shape	Circle
Elements	46
Bar Width	59
Bar Height	203
Rounded Corners	True
Size Mode	Fixed

5.3 Constructing the Evaluation Dataset

After setting up the rating system and collecting expert opinions, the next step was to convert the saved ratings into a well-organized dataset for analysis. Instead of saving images, the system only kept the settings used to make them. Since any image can be made again from these settings, this method makes the process clear, easy to track, and accurate. Each rating is directly connected to the exact settings that created it.

In this way, the dataset is not just something added later but is part of how the generator is built. The way the database is organized matches how the system works inside.

5.3.1 Structure of the Dataset

To make things clear and easy to understand, all collected JSON files were reorganized into a standard format with three clear sections: identity and context, radial parameters and polygonal parameters.

Table 5.2 Identity and Context Variables of the Evaluation Dataset

Identity	Variable	Description
session	rater_group	self / communication_office
	session_id	Unique session identifier
	timestamp	Time of evaluation
contex	mode	radial / polygonal
	shape	Main geometric category
	polygon_type	polygon subtype (if applicable)
rating	rating_label	unsuitable – suitable
	rating_numeric	Numerical scale (1–5)
analysis fields	rotation	Rotation shape value
	density_proxy	elements for radial sides for polygonal

These variables allow filtering by evaluator group, generative mode, or rating level without mixing parametric logic.

Table 5.3 Polygonal Mode Parameters (NULL for Radial Records)

Polygon	Parameter	Notes
star **gear**	sides	Number of polygon sides
	innerRadius	Inner radius (star / gear)
gear	topLand	Top segment width (gear only)
	bottomLand	Bottom segment width (gear only)
Polygon **star** **gear**	cornerRadius	Corner rounding value
	roundedCornersMask	Specifying which corners are rounded

Table 5.4 Radiall Mode Parameters (NULL for Polygonal Records)

Radial	Parameter	Notes
structure	elements	Number of radial elemets
element geometry	barWidth	Primary radial width
	barWidth2	Trapezoid width
	barHeight	Element height
edge refinement	cornerRadius	Corner rounding value
visual boundary	strokeWeight	Stroke thickness
	strokeMin	only concentric mode
	strokeMax	only concentric mode
Concentric Extension	concentric_rings	Number of concentric repetitions
	concentric_angleOffset	Rotational offset between rings
	concentric_sizeMode	inward / outward / fixed

Importantly, any settings that did not matter for a certain mode were clearly set to NULL. This prevents incorrect links from appearing during analysis and keeps the results easy to understand. This separation shows that the generator works in two ways: radial and polygonal shapes use different geometric rules, so they need to be analyzed differently.

5.3.2 Filtering Strategy

Before starting the analysis, the dataset was filtered to match the evaluation goals. Two main filtering strategies were used:
Mode-based filtering; radial and polygonal samples were analyzed separately because their shapes differ.
Rating-based filtering; to find patterns, we paid special attention to ratings 4 and 5 (Suitable and Very Suitable), as these indicate setups that align with the institution's branding guidelines.

This filtering keeps all the data but highlights the setups that are most suitable, making it easier to see new trends. During data collection, this shift moves the evaluation from subjective judgment to reproducible analytical groups. The dataset itself becomes

a design tool. It encodes an institutional judgment, parametric structure, and evaluates the levels used to judge if something is suitable. By viewing evaluation as qualitative commentary, the system turns professional design expertise into structured, comparable data. This organized dataset enables the identification of patterns, connections, and differences between evaluator groups and design methods in the next step.

5.4 Analysis of Data and Emerging Patterns

After the evaluation data set was created, the analysis was divided into three main parts: the Office of Communications' assessments in the radial mode, the Office of Communications' assessments in the polygonal mode, and the self-assessments in both modes. The results of the different groups were then compared. This setup makes things clear and useful. The Office of Communications provides an expert, official view, while the self-assessment offers a practical, perhaps not entirely objective view from the author. This division helps identify consistent patterns, shows differences in how things are assessed, and checks whether the system aligns with the institution's goals. Rather than looking for the best shape, the analysis looks for consistent ranges of values, good results, patterns in complexity, and differences between the radial and polygonal modes.

5.4.1 General Distribution Overview

Here is an analysis of all the data, namely the main parameters compared with low- and high-ranking cases. This is the first insight into the differences or similarities, the first patterns. To find out whether the shape affects how people rate its appearance, we changed the ratings for each shape and then compared them as percentages. The results show big differences between the types of shapes (Fig. 5.3).

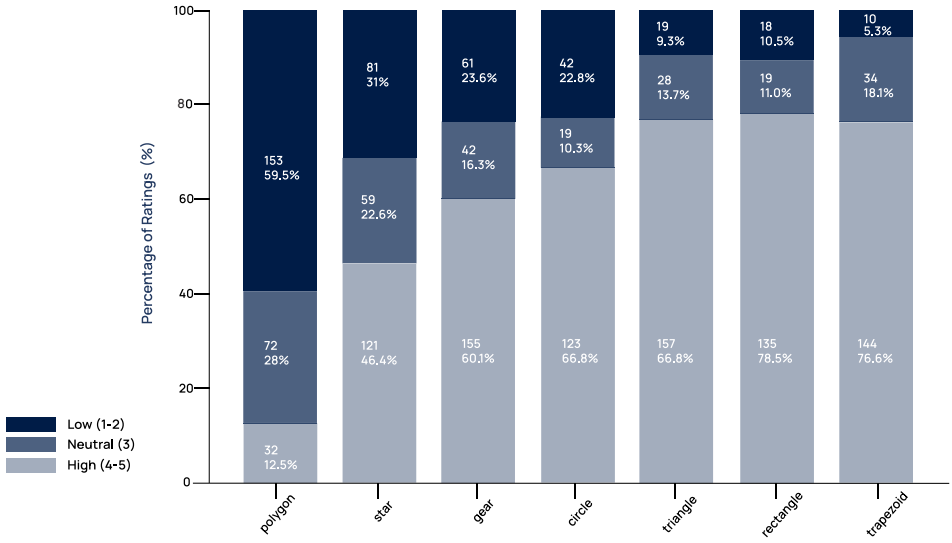


Fig. 5.3 Rating Distribution by Shape Category

The polygon shape received the highest ratings, indicating it was the most frequently shown shape in random generation: 153 high (59.5%), 72 neutral (28.0%), and 32 low (12.5%). That doesn't mean it is the preferred shape; it will only appear in comparisons of high ratings. The star shape had mixed feedback: 81 high (31.0%), 59 neutral (22.6%), and 121 low (46.4%), showing people had different opinions about its branding. The gear shape mostly gotnegative ratings, with 155 low (60.1%), 61 high (23.6%), and 42 neutral (16.3%).

The basic shapes mostly got low ratings. The circle had 42 high ratings (22.8%) and 123 low ratings (66.8%). The triangle had 19 high (9.3%) and 157 low (77.0%). The rectangle had 18 high (10.5%) and 135 low (78.5%). The trapezoid had the lowest results, with only 10 high (5.3%) and 144 low (76.6%). The data suggest that shape type is the main factor in how good people think it looks in this test.

5.4.2 Failure Pattern Analysis (Low ratings)

To identify patterns in negative evaluations, the analysis focused only on designs rated 1 or 2 (Low). By focusing on these, it was possible to spot recurring structural features. The goal was to determine whether negative ratings occur randomly or are linked to specific parameter settings.

Out of 1,524 samples, 867 received a Low rating (56.9%). In this group, certain structures stand out. Most low-rated designs (64.5%, or 559) were made in radial mode, while 35.5% (308) were made in polygonal mode. This shows that radial designs are more likely to get negative ratings. Two main factors stand out among the low-rated designs. High density (above the median) is found in 59.1% (512) of them. The same percentage also has high structural complexity, meaning they have more elements and sides than average. So, almost three out of five rejected designs are more complex than most. The overlap between high density and high complexity suggests that too much visual detail is a key reason for failure. Looking at the data by groups from lowest to highest density supports this pattern. As density increases, the chance of a low rating increases. This steady increase shows a strong link between density and the chance of rejection.

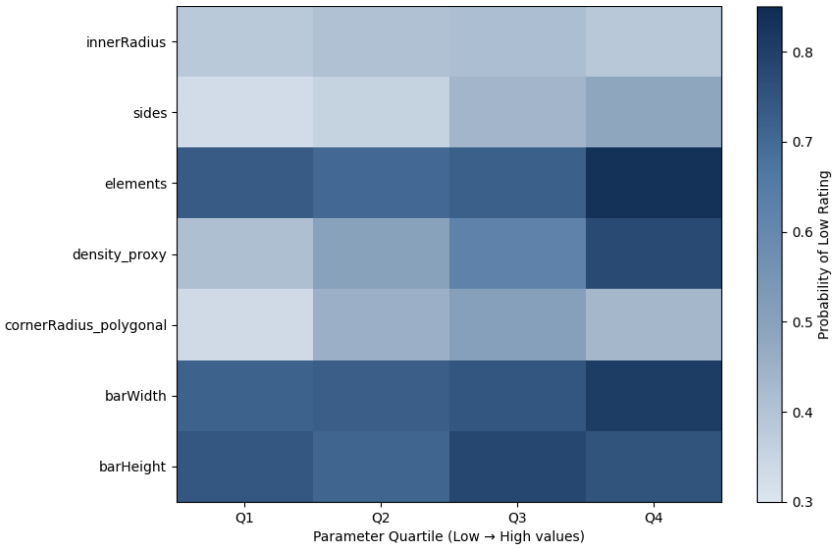


Fig. 5.5 Low Rating Probability Across Parameter Quartiles

To talk about low rating probability (Fig. 5.5), the scale that goes from 0.3 to 0.8 is introduced. The numbers 0.3–0.8 represent probabilities (30%–80%) of receiving a Low rating. The scale does not start at 0 because, in this dataset, the lowest observed probability of a Low rating in any quartile is about 30%. Likewise, it does not reach 1 (100%) because the highest observed probability is about 83%. The heatmap, therefore, shows the true range of failure probabilities observed in the data. Lighter cells (around 0.3–0.4) mean that roughly 30–40% of designs in that quartile received a Low rating, while darker cells (around 0.7–0.8) mean that about 70–80% did. The scale simply reflects the actual variation present in the dataset. A similar, but smaller, trend appears in the number of sides. Designs with the fewest sides have about a 33% chance of a Low rating, while those with the most sides have about a 48% chance. So, more subdivisions slightly raise the risk of failure.

The number of elements is associated with higher failure rates across all quartiles, with about 70% to 83% of designs in the higher quartiles receiving a Low rating. This shows that repetition and more radial elements are strongly tied to rejection.

In contrast, changing the inner radius does not show a clear pattern. The chance of a Low rating remains about the same across all groups (about 38–41%), indicating that changing only the inner radius does not predict a negative rating. This suggests that designs fail due to overall complexity, not just a single shape change. Radial thickness also supports this idea. Designs with the thickest bars have about an 80% chance of a Low rating, and bar height also stays high (about 70–78%). Thicker and more noticeable bars make the design feel too crowded.

Concentric rings make negative ratings more likely, but they are not the only reason. 38.5% of low-rated designs (334) use concentric rings. When these rings are used, the chance of a Low rating jumps: 82.9% of all concentric designs get a Low rating, compared to 47.5% when there are no concentric rings. If the design also uses changing sizes, the risk of failure increases further. For example, designs with Outward scaling have almost 89% Low ratings. Other structural types also play a role, though less often. Gear shapes make up 17.9% (155) of low-rated designs, and

star shapes make up 14.0% (121). These shapes seem to raise the chance of rejection, especially when paired with high density and complexity. In summary, low-rated designs tend to share certain features: radial geometry, high density, lots of subdivisions, and many elements. The data suggest that designs are rejected primarily because of excessive visual complexity. It is not just one factor, but a combination of several complexity-related variables that makes a design overwhelming.

5.4.3 Communication Office – Radial Shapes

The first analytical step focuses on the Communication Office ratings for radial constructions. Only ratings of 4 (Suitable) and 5 (Very Suitable) are included in this phase, as the goal is to identify configurations that demonstrate institutional acceptance rather than borderline or rejected outcomes.

Table 5.5 Distribution of Highly Rated Radial Shapes (Institutional, Ratings 4–5)

Shape	Count	Avg. Elements	Min.	Max.
Circle	26	33.5	10	68
Triangle	12	37.53	10	70
Rectangle	14	28.64	6	65
Trapezoid	3	31	10	45

The circle clearly dominates in institutional approval.
The trapezoid is marginal, appearing rarely among high-rated outputs. The triangle shows the highest average number of elements. Rectangles operate at slightly lower density levels. The dominance of circular symmetry suggests that radial clarity and geometric neutrality align most closely with Politecnico's visual identity. Shapes that introduce directional or asymmetrical tension appear less consistently suitable.

Table 5.6 Density (Elements) and Rating Differentiation of Radial Shapes

Ratings	Avg. Elements
4	35.08
5	28.9

A subtle but important distinction emerges: Very Suitable (5) shapes have a lower average density than Suitable (4) shapes. This suggests that while moderate complexity is acceptable, excessive repetition begins to weaken perceived suitability. The Communication Office appears to favor: controlled repetition, spatial clarity and moderate visual rhythm rather than maximal generative intensity.

Table 5.7 Bar Width and Vertical Extension by 4 and 5 Institutional Ratings

Shape	Avg. barWidth	Avg. barHeight
Circle	39.16	191.5
Triangle	36.87	199.6
Rectangle	33.42	146.6
Trapezoid	36.66	189.6

Triangles and circles tend to maintain greater radial extension, reinforcing their strong geometric presence. Rectangles remain visually more compact. However, variation in barWidth does not strongly correlate with higher ratings. This suggests that density and symmetry matter more than proportional thickness. This parameter likely functions as a fine-tuning control rather than a decisive variable.

Table 5.7 Average corner rounding in radial shape by institutional evaluation.

Shape	Avg. Corner Radius
Circle	9.8
Triangle	7.75
Rectangle	4
Trapezoid	8

Unlike density, corner rounding in radial shapes shows a more selective pattern. Circles tend to tolerate higher cornerRadius values, reinforcing smooth visual continuity. Rectangles and triangles display moderate rounding, likely enhancing structural clarity without softening the geometry excessively. Trapezoids often appear with minimal or zero rounding, indicating that their structural asymmetry may already provide sufficient dynamism without additional softening.

Concentric structures are rarely present in high-rated radial shapes:

```
Fixed: 3          Inward: 1       Outward: 1
```

Most highly rated radial shapes are single-layer constructions. This indicates that institutional preference leans toward simplicity over layered complexity. Concentric logic, while technically robust, does not appear central to perceived suitability.

5.4.4 Communication Office – Polygonal Shapes

Polygonal constructions present a more varied and expressive field. Unlike radial shapes, here internal structure and edge articulation play a stronger role.

Table 5.8 Polygon type Distribution (Institutional Ratings 4–5)

Shape	Rating 4	Rating 5	Total	Number of Sides
Polygon	86	44	130	8
Star	22	37	59	14
Gear	24	12	36	15

The regular polygon appears most frequently. The star receives proportionally more 5 ratings than 4 ratings. The gear receives more 4 than 5 ratings. This suggests: The regular polygon is institutionally safe. The star is visually expressive and capable of achieving excellence. The gear is technically acceptable but less likely to be judged “very suitable.” The regular polygon stabilizes around eight sides, suggesting that moderate segmentation aligns

with the brand's geometric language. Stars and gears operate at higher segmentation levels, but only stars convert this complexity into higher 5 ratings. Because the polygonal interface constrains the minimum number of sides to 5, the minimum value is a design constraint rather than an evaluative outcome; analysis therefore focuses on the distribution (median/mode) and how side counts relate to rating differences (4 vs 5). This indicates that complexity must be expressive, not merely mechanical.

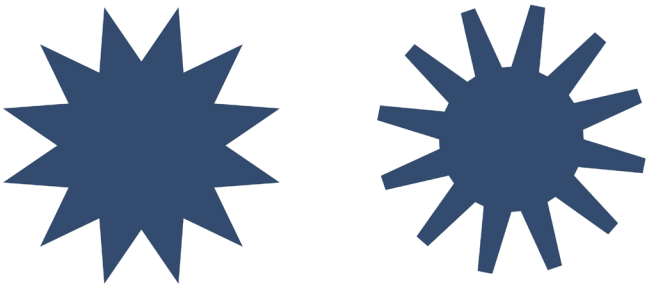


Fig. 5.6 Comparison of Average Inner Radius Values for Star and Gear Shapes by Institutional evaluation.

Higher inner radius values soften angular aggressiveness. Stars with larger inner radii appear more balanced and less visually aggressive. This visual comparison clarifies how the innerRadius parameter structurally affects form: higher values increase indentation depth and elongate spikes, while lower values create denser, more compact configurations (Fig 5.6). The figures therefore make explicit the geometric contrast between star and gear constructions within the evaluated dataset.
A particularly strong pattern appears in polygonal corner radius:

```
Gear (Rating 5) → higher cornerRadius than Rating 4
Star (Rating 5) → higher cornerRadius than Rating 4
Polygon → relatively stable moderate rounding
```

This indicates a clear institutional preference: softer edges increase suitability. Sharp, rigid geometry appears less aligned with the Politecnico identity. High-rated shapes frequently display consistent rounding across all vertices rather than selective rounding.

This suggests that: Visual coherence is preferred over partial ornamentation. Consistency supports brand clarity. From both radial and polygonal analysis, several consistent tendencies emerge:
Moderate density is preferred over maximal repetition.
Circular symmetry performs best in radial mode.
The star performs best among expressive polygonal shapes.
Softened geometry (higher cornerRadius) increases suitability.
Concentric complexity does not strongly correlate with higher ratings. Consistency across vertices is valued over selective variation.

5.4.5 Self Evaluation – Radial Mode

The self-evaluation dataset includes 205 samples, 108 radial and 96 polygonal. Although relatively modest in size, the dataset was intentionally balanced against the number of evaluations per individual from the communication office group in order to maintain structural comparability across rater groups.

Table 5.9 Distribution of Highly Rated Radial Shapes (Internal Ratings 4–5)

Shape	Count	Avg. Elements	Min.	Max.
Circle	16	45.4	14	79
Triangle	4	40	6	66
Rectangle	4	39.3	13	77
Trapezoid	7	46.1	10	79

Circle is again dominant, but not as strongly as in the Communication Office results. Trapezoid performs significantly better in self-evaluation (7 high ratings) compared to the Communication Office. Rectangles and triangles remain secondary forms.

This suggests a slight difference in aesthetic tolerance: the self-evaluation appears more open to structural experimentation (especially trapezoids), whereas the Communication Office favored circular clarity more strongly.

The preferred density in self-evaluation is moderately high (around 40–46 elements). Self density averages are slightly higher than Communication Office medians. Extreme minimum values (10–14) still appear in high ratings. Very high density (79) is accepted in some cases. Density is not rejected outright when high, but likely interacts with shape type.

Table 5.10 Bar Width and Vertical Extension by 4 and 5 Internal Ratings

Shape	Avg. barWidth	Avg. barHeight
Circle	49.06	178.06
Triangle	24.55	128.75
Rectangle	41.25	232.5
Trapezoid	45.29	191.7

Average Width is 44. Trapezoid barWidth2 average is 56.85. Bar-Height is significantly higher for trapezoid (191.7) and cir cle (178). Self-evaluation seems to prefer:
Relatively tall radial elements;
Medium-to-wide bar widths;
No strong restriction on barWidth range;
Unlike density, barWidth variation appears broad and permissive.

Table 5.11 Average corner rounding in radial shape by internal evaluation.

Shape	Avg. Corner Radius
Circle	10.5
Triangle	7.16
Rectangle	7.25
Trapezoid	0

Circles tolerate higher rounding. Trapezoids were accepted without rounding. Rounding is not uniformly required. This suggests rounding enhances clarity in simple geometries but is not necessary for more complex ones.

5.4.6 Self Evaluation – Polygonal Mode

Table 5.12 Polygon type Distribution (Internal Ratings 4–5)

Shape	Rating 4	Rating 5	Avg. Sides (4)	Avg. Sides (5)
Polygon	16	7	7.6	9.5
Star	20	2	14.7	17.5
Gear	23	2	14.3	18

Gear dominates numerically, but mostly in rating 4.
Polygon type receives the highest number of 5 ratings (7).
Star receives moderate acceptance but fewer top scores.
This indicates that gear is tolerated; polygon is preferred when excellence is perceived; star is visually strong but less consistently aligned with branding. Higher ratings (5) tend to correspond to: slightly higher side counts in polygon and star; more complex geometry for 5 than 4. This is subtle but consistent.

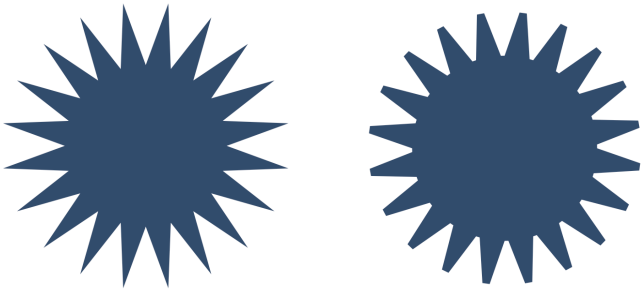


Fig. 5.7 Comparison of Average Inner Radius Values for Star and Gear Shapes by Internal evaluation.

Gear innerRadius is much larger for rating 5. Star innerRadius slightly decreases. This suggests: For gear, higher separation between outer and inner radius improves perceived suitability. For star, subtle balance matters more than extremity (Fig. 5.7).

Table 5.11 Average corner rounding in polygonal shape by internal evaluation.

Shape	Avg. Corner Radius (4)	Avg (5)
Polygon	11.3	14
Gear	6.7	-
Star	8.5	6

Polygon benefits from rounding when rated 5.
Star slightly reduces rounding in 5 ratings.
Gear does not rely heavily on rounding.
This confirms that rounding is shape-dependent.

5.4.7 Cross Comparison of Evaluator Groups

After analyzing radial and polygonal results separately for each evaluator group, a cross-comparison was conducted to identify structural differences between institutional and individual judgments. The comparison focuses exclusively on ratings 4 and 5, since these represent shapes considered suitable for use within the Politecnico di Milano visual identity. A first structural difference appears in the distribution of positive ratings across generative modes.

Rater Group	Polygonal (4–5)	Radial (4–5)
Communication Office	225	58
Self	70	31

The Communication Office shows a strong preference for polygonal constructions. More than three quarters of their positive ratings fall within polygonal mode. The self-evaluation remains more balanced between radial and polygonal outputs. This suggests that institutional judgment aligns more strongly with geometric constructions that reflect structured, controlled polygonal systems, while the individual evaluator allows more variation across generative logics. Radial compositions were compared using the parameter elements (used as density proxy). Average number of elements (ratings 4–5 only):

Rater Group	Average Elements
Communication Office	33.24
Self	44.10

This implies that excessive repetition reduces institutional suitability. Complexity is acceptable, but clarity is decisive. The self-evaluation tends toward higher complexity and denser radial structures. This suggests a difference in tolerance for visual saturation. Institutional evaluation appears to prefer clarity and legibility, while individual exploration accepts or even favors increased structural richness. Both groups favor circles. Communication Office marginalizes trapezoids. Self-evaluation grants trapezoids significantly more high ratings. Circles represent geometric neutrality, they consistently align with the institutional identity.

Polygonal constructions were compared using the number of sides. Average number of sides (ratings 4–5 only):

Rater Group	Average Sides
Communication Office	10.84
Self	12.63

Regular polygons dominate numerically. Stars receive proportionally more rating 5 than 4. Gears remain technically acceptable but rarely excellent. The self-evaluation leans toward higher geometric complexity, while the Communication Office remains closer to controlled, mid-range polygonal configurations. Both evaluators exhibit a subtle higher ratings correlate with slightly higher side counts. However, Communication Office stabilizes around mid-range segmentation (10–11 sides).

Self-evaluation accepts slightly higher segmentation (12–15 sides). Across both generative systems, a consistent divergence can be observed: The institutional evaluator favors structural moderation. The individual evaluator gravitates toward increased density and complexity. This does not indicate disagreement in qualitative direction, both groups positively evaluate many of the same shape families, but it reveals different thresholds of suitability. The Communication Office appears to operate with a narrower

tolerance band, aligning more closely with branding consistency and reproducibility across media. The self-evaluation reflects a broader exploratory range within the same parametric space.

The cross-comparison confirms that evaluation is not merely descriptive but structurally revealing. Although the generator is rule-based and deterministic, the ratings demonstrate that suitability emerges from parameter relationships rather than from rule compliance alone.

Institutional	Individual
Moderate density	Higher density tolerance
Geometric restraint	Structural richness
Softer edges	Shape-dependent rounding
Stable segmentation	Slightly higher complexity
Narrow tolerance band	Broader exploratory range

Even though the generator follows set rules and always works the same way, merely following the rules does not guarantee results that are a good fit. Whether something is suitable depends on how the settings work together, how crowded things are, how shapes are balanced, and how the edges are treated. Group judgment checks if the results match brand stability needs. Personal judgment helps spot where creative differences are still okay. This difference becomes helpful in the next step, when groups of settings are compared to types of reviewers. This shows not only what is suitable, but also who it is suitable for.

06

Future Development and Ethical Considerations

The current version of the system uses a deliberately small dataset. The testing phase involved a small group: the Communication Office and the author. While this means the results cannot be widely applied, it yields a focused set of data informed by knowledge of the institution's brand. This limitation should be clearly stated. However, it does not mean the system is weak; instead, it shows that this is a carefully controlled test phase. The dataset now serves as a diagnostic tool for observing the behaviour of the random engine and as a testing ground to spot repeating patterns in the settings or where repetition emerges. It is a starting point for assessing how well the designs created match the university's look. Early results show that shapes like polygons and circles, especially those that are evenly arranged and closely match the current brand, get higher ratings. This is expected because the brand already uses these types of shapes. On the other hand, less regular or more unusual shapes received lower scores. If the system were to track these results without careful tuning, it would produce only a small set of top-rated designs. This would make the designs less diverse and more repetitive. This already poses a problem for the implementation and development of smart random buttons. Should the system optimize for current preferences, or should it preserve the exploratory capacity to refine elements that do not perform well? The current dataset, although limited in size, is significant because it reveals where generative loops occur and where bias in evaluation may occur. It allows for the identification of structural tendencies before scaling the system to a larger participant base. Adding more university departments to the dataset, including those with and without design experience, would introduce greater variety in evaluation. Departments not involved in branding might be more open to unusual designs, which could

help the system create a wider range of results. However, scaling the dataset introduces new methodological challenges. Looking at lots of settings and big dataset is a problem for designers. Without the right tools or help from data experts, there is a real risk of misunderstanding the results or making mistakes. At this stage, the project establishes a foundational framework based on a structured parameter system and a rating-based feedback mechanism. There is a first set of data showing how the system tends to create designs. Future development must therefore focus on optimizing the generative logic and systematically expanding the evaluation infrastructure.

The next sections explain how learning from user choices, adding machine learning, and growing the system in the institution could develop from this starting point.

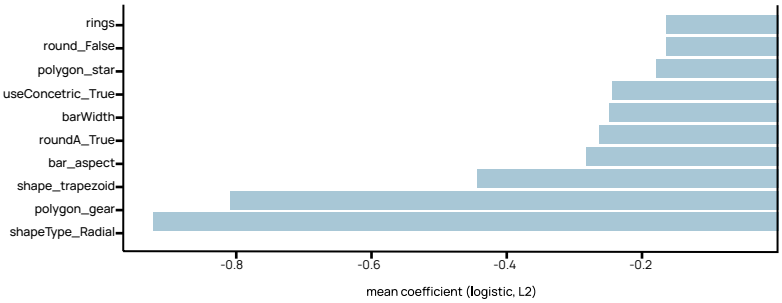
6.1 Smart Random and System Adaptation

Before introducing machine learning to the system, it is important to assess how much we can improve it without it. The generator was first designed as a random engine that picks values for its settings from set ranges and creates a fixed visual result from those settings. Here, "random" does not mean messy or without rules; it means selecting values at random from within set limits. Each setting (shapeType, polygon, sides, useConcentric, barWidth, or cornerRadius) is chosen from a set group of options, and once picked, the generator makes a steady result.

However, picking values randomly in this way has a basic problem. It assumes that all possible settings are equally good. The study showed this is not true. Some designs were rated better than others. For example, shapes like regular polygons and circles received higher ratings, while very complex shapes, stretched shapes, and some rounded shapes received lower ratings.

This pattern shows that some settings are clearly preferred over others. The generator does not treat all designs equally; some settings fit people's expectations much better.

(A) Positive coefficients (features increasing probability of high ratings)



(B) Negative coefficients (features increasing probability of low ratings)

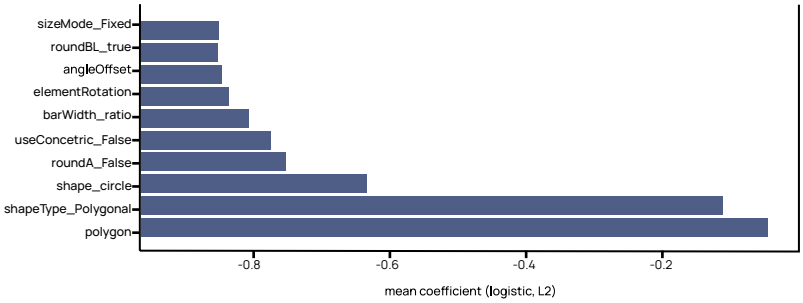


Figure 6.1 Logistic Regression Feature Coefficients Associated With High and Low Suitability.

Each bar represents a one-hot encoded feature used by the logistic regression model. Coefficients indicate the direction and relative strength of association with high suitability ratings. Positive values increase the probability of high ratings, while negative values increase the probability of low ratings. Larger absolute values indicate stronger influence (Fig. 6.1). The coefficient patterns confirm that parameter values do not contribute equally to perceived suitability. These empirical tendencies form the statistical foundation for the Smart Random weighting mechanism.

If the generator keeps sampling randomly, two problems arise: high-preference configurations appear less often, and low-preference areas are explored just as much as high-preference ones.

This leads to wasted effort and repeating designs that do not look as good. During testing, another problem showed up: looping. Looping occurs when the generator keeps generating designs with different settings that look almost the same. This usually happens when many different settings produce similar-looking

results. Increasing the number of rings beyond a certain threshold increases visual clutter rather than structural richness. Extreme barWidth ratio values produce elongated elements that visually collapse into noise. Certain combinations of rounding flags lead to similar softened geometries.

Randomly picking settings does not notice when designs look the same. It treats changes in settings as if they always make the designs look different. This shows that there is a need to make changes not just to match what people like, but also to keep the designs varied.

6.1.2 Smart random adaptation without ML

Evaluation Dataset → Parameter Frequency Analysis → Weighted Sampling Table → Generator

The first stage of adaptation does not require a trained statistical model. Instead, the plan is to use the results from the evaluation dataset to adjust the generator's sampling distributions. Of course, after an expanded database and new tests, a wider range of different cases and parameters must first be collected to avoid the system becoming too rigid. This approach, called "Smart Random Selection," keeps the process random while increasing the probability of selecting stronger regions of the parameter space. For example, instead of sampling shape types uniformly:

```
// Original uniform sampling
gui.shapeType = random(["Radial", "Polygonal"]);
```

The distribution can be changed based on how often each preference is seen:

```
// Data-informed biased sampling
gui.shapeType = randomWeighted([
  { value: "Polygonal", weight: 0.65 },
  { value: "Radial", weight: 0.35 }
]);
```

Similarly, high rings counts can be sampled with lower probability:

```
gui.rings = randomWeighted([
  { value: 2, weight: 0.30 },
  { value: 3, weight: 0.30 },
  { value: 4, weight: 0.25 },
```

```
  { value: 5, weight: 0.10 },
  { value: 6, weight: 0.05 }
]);
```

Numeric ranges can also be constrained:

```
// Restrict extreme aspect ratios
gui.barHeight = floor(random(20, outerRadius * 0.7));
```

This approach does not need the model to make predictions. It uses simple statistics from the evaluation dataset. Parameters linked to high ratings are chosen more often, while those linked to low ratings are chosen less often. The generator grammar stays the same. No parameters are taken out. The structure does not change. Only the chances of picking each parameter are adjusted.

Smart Random (Independent Weighting):

```
polygonType → weighted
sides → weighted
cornerRadius → weighted
```

ML-based Learning (Interaction Modeling):

6.1.3 Advantages and Limitations of Smart Random

Clearly, the first advantage of this random-tuning improvement is transparency: everything is explicit and manually adjustable in the code itself. And if it is more complicated to improve the program in this way because it is limited, there is no dependence on machine learning and new code infrastructure for implementation. It is also very convenient that designers keep full control over the constraints. From a system design perspective, this represents an adaptive layer of low complexity. It introduces a response to empirical findings without increasing the architectural burden. Despite this convenience, even without implementation, it is possible to identify potentially larger problems. Just by analyzing the existing data, it can be seen that there is

a probability that individual parameter values can be increased, but it cannot easily model parameter interactions. For example, it cannot detect that polygon=”star” can only work well when sides ≤ 8 and polygonRounded=false. Such conditional patterns require learning common feature relationships.

Heuristic bias can gradually reduce diversity. If strong configurations are sampled more often without controlling for diversity, the generator can converge towards visually homogeneous outputs. This is the case with the database we have. If the data system were to expand to other departments by adding new evaluators, the heuristic weights would need to be manually updated as new data arrives. There is no automatic adaptation mechanism. In short, Smart Random improves sampling efficiency but does not fundamentally learn the structure of preference.

Smart Random is a middle step between completely random generation and learning from data. It shows that the system can improve by using guided random choices informed by feedback. However, as there are more settings that interact in more complex ways, simply adjusting things by hand is not enough. Then, the system needs a way to learn and assess how different settings interact. This transition motivates the next section because Smart Random reshapes probabilities, whereas machine learning estimates them.

6.2 Preference Learning and ML Integration

The current system works as a structured generator that uses set rules. It sets clear limits for design using variables such as shapeType, polygon, rings, elements, and barWidth, along with fixed rules for how things are displayed. The system creates visual forms from clear settings, not from random noise. Each result can be saved with parameters. Such a system is suitable for connection with machine learning from the start. A term we already encounter every day in the context of AI is returned here by the ML system without adaptation. ML was defined in the 1950s by AI pioneer Arthur Samuel as “the field of study that gives com-

puters the ability to learn without explicitly being programmed." Simply put, here the program would learn from a large database of preferences through parameters.

Instead of following step-by-step instructions, the system analyzes data patterns to make decisions. For example, it is like when Spotify suggests songs you may like. With this setup, the best way to use machine learning is to learn what people like within the range of settings, instead of replacing the generator itself. So the question is not "Can ML generate shapes?" but "Can ML predict which parameter configurations are likely to be evaluated as suitable?"

This work is suitable for a reward-model approach. In machine learning, this method aligns with human preferences by encouraging good behaviors and discouraging unwanted ones (Fig. 6.1).

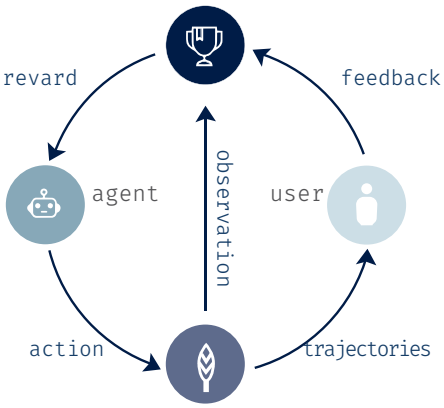


Fig. 6.1
Illustration of reward-model approach.

A reward model is trained on data about what people prefer, so it can predict which outcomes a person would choose. The reward model usually acts as a judge, giving feedback to the system being trained. This back-and-forth helps the system get better (Bai et al., 2019). The goal is to learn a function:

```
p(high| params)
```

where: high=1 if rating 4 which represents the probability that a given configuration will receive a high rating (e.g., rating ≥ 4), and function is learned from logged evaluation data. This shift maintains authorship and control. It also aligns with Shneiderman's

Human-Centered AI framework, which holds that high automation should work alongside strong human control rather than replace it (Shneiderman, 2020).

Machine learning models need input that is a list of numbers of the same length. But the generator produces different kinds of parameter objects that also contain words: categorical variables (strings), numerical variables (integers and floats), boolean flags, and conditionally defined fields. To train a model to make predictions, the parameter object must be converted into a list of features in a fixed order. This process is called a feature engineering pipeline. Category values like `shapeType` (`"Radial"`, `"Polygonal"`) cannot be used as words. Instead, they are encoded using one-hot encoding, which creates a column for each category with values of 0 or 1. This method converts categorical attributes described by words into a binary vector format, thereby preventing the introduction of arbitrary ordinal relationships that could arise from simple numerical assignments. (Matos et al., 2022)

Pyton
```
# Example categorical column
shapeType = ["Radial", "Polygonal", "Radial"]
```

```
# After one-hot encoding:
shapeType_Radial     shapeType_Polygonal
                1                       0
                0                       1
                1                       0
```

This stops the model from thinking the categories (e.g., Radial = 0, Polygonal = 1) have a natural order or ranking, which could create false patterns. Number features like barHeight and rings can have very different ranges. Logistic regression works by multiplying each feature by a weight and summing the results. Because of this, large-range number variables can have too much influence on the model's results, making the weights unfair and reducing how well the model predicts. If barHeight ranges from 20 to 400 but rings only range from 2 to 5, the larger numbers can take over just because they are larger. To fix this, numerical variables are standardized, which means:

$$x_{\text{scaled}} = \frac{x - \mu}{\sigma}$$

where $\mathbf{x}$ is the original feature value, μ is the mean of that feature computed on the training data, σ is the standard deviation of the feature computed on the training data. This standardization process scales all numerical features to a common range, usually with an average of zero and a standard deviation of one, so features with larger ranges do not dominate and the model can learn more easily (Renzo et al., 2025).

Because the generator has different paths, some parameters are missing in certain cases. For example, polygon fields are not available in radial mode. During training, missing values are imputed with the mean for numerical features and the mode for categorical features. The same method must be used when making predictions. If the steps differ between training and prediction, the results will be incorrect. All of these data preparation steps are performed in Python during training, and the required values (such as averages, standard deviations, and category lists) are saved for use in JavaScript.

6.2.1 Training the Preference Model

Training the preference model is not just a technical task. It is the main building block that enables flexible generation methods such as reward-guided sampling and possibly Bayesian optimization. Both methods use the same trained prediction function. They are not different systems, but ways of using the trained model. The objective of training is to estimate a function:

$$R(\mathbf{x}) = P(Y = 1 \mid \mathbf{x})$$

where $\mathbf{x}$ represents a vector of generative parameters and Y = 1 indicates that a configuration receives a high rating (rating ≥ 4). The model therefore estimates the probability that a given parameter configuration will be judged suitable. This is a supervised learning task where examples with known parameters and ratings are used to teach the model how features relate to preferences. The training process follows a precise sequence. First, all JSON files are loaded and combined into a table, with each rated setup

```
JSON Ratings
    ↓
Feature
Engineering
    ↓
One-Hot Encod-
ing + Scaling
    ↓
Logistic
Regression
Training
    ↓
Reward Function
R(x)
    ↓
JavaScript
Deployment
```

as a row. The nested settings are split into clear feature columns. Each file is marked with the name of the rater so that any bias can be checked later.

Next, the original five-point rating scale is converted into a binary target variable called high. This changes the rating scale into a simple sorting problem. Although a more detailed method is possible, this simpler approach is easier to set up and better suited to finding likely good setups. Since the settings include both text and numbers, but machine learning models require numbers, text and true-or-false features are split into separate columns, and number features are scaled to match.

This data preparation step helps the model learn true patterns rather than being influenced by differences in scale. The validation strategy is equally important. Using GroupKFold, entire raters are excluded during validation folds. This ensures that the model learns generalizable preference patterns rather than memorizing individual evaluator tendencies. Rater-aware validation is particularly relevant in interactive systems where user feedback may vary substantially, a challenge noted in interactive machine learning research (Amershi et al., 2014). The approach also aligns with human-centered AI principles that emphasize robustness across user groups (Shneiderman, 2020).

Logistic regression is chosen as the main predictive model. While more complex models might perform better, logistic regression is easier to understand because each number in the model matches a setting. This supports openness and trust. It is also simple to run on computers and can be easily used in JavaScript as a dot-product-plus-sigmoid function.

A key finding from the current testing phase concerns how to interpret predicted probabilities. For example, if the model predicts `p(high) = 0.5`, it does not mean there is exactly a 50% chance of getting a high rating. Instead, this number should be compared with other predictions. In the dataset, the number of high ratings given by different people varied a lot (from about 0.07 to 0.43). In this situation, predicted probabilities above 0.5 indicate results higher than most learned preferences. This shows an important idea: the model's output should be seen as a ranking

score, not as an exact measure of truth. The accuracy of these scores can be improved, but for creating results, the order of the scores matters more than the exact probability values.

After training, the model acts as a scoring function for the parameter space. Rather than swapping the parametric generator for a generative neural model, the system keeps clear constraints and uses the learned function to explore the space more efficiently.

6.2.2 Implementing the Model in JavaScript

When the model is trained in Python, the system is converted to JavaScript with a predefined formula.

$$p(high) = \sigma(w \cdot x + b)$$

This can be implemented directly:

```
function sigmoid(z) {
 return 1 / (1 + Math.exp(-z));
}
function modelPredict(features) {
 let z = bias;
 for (let i = 0; i < weights.length; i++) {
   z += weights[i] * features[i];
 }
 return sigmoid(z);
}
```

Here, the features are the processed input values, weights, and bias, which come from training in Python. There is no need for any special machine learning software, server, or neural network tools. Now, in the generator, we have a command for random generation:

```
randomizeSettings();
render();
```

With preference learning, generation will become:

```
randomizeSettings();
let x = featurize(gui);
let p = modelPredict(x);
if (p ⩾ threshold) {
 render();
```

```
randomizeSettings();
let x = featurize(gui);
let p = modelPredict(x);
if (p ⩾ threshold) {
 render();
}
```

This approach introduces a reward-based search mechanism rather than a full reinforcement learning loop. The reward model estimates how good a choice is, such as a parameter setup, without running a full test. Rather than replacing the generator with a neural network that generates new data, the system maintains its rule-based architecture and uses the reward model to guide the search.

This method avoids the issues that come with using fully generative table-based models like CTGAN on small datasets. CTGAN is a deep learning model made to generate synthetic tabular data. It uses a conditional generator and special normalization techniques to handle different data types and distributions, often for data augmentation or privacy (Khosravi et al., 2024). With around 1,300 samples and many categories, generative models might just memorize the data or create impossible combinations. Using a reward model with search keeps the rules and structure set by the designer clear.

The model does not learn the visual part of the program; it finds patterns between parameter settings and how people rate them. For example, shapes with more sides might receive positive scores, while high ring values might receive negative scores. Certain width-to-height ratios could make a design feel cluttered. These scores reflect what the model has learned about preferences in the design space. The model changes the likelihood of selecting certain options under the current rules. This method, in which a reward model guides search in a rule-based system, transforms the problem into a more traditional reinforcement learning setup in which actions are chosen in an uncertain environment (Mu et al., 2024). Adding preference learning to a parametric generator changes the process from random selection to a smarter search. The training creates a reward model that predicts the likelihood

of a high score, which is checked by multiple people and adjusted using a simple statistical method to ensure clarity. The system only changes the program's random option, not the program itself.

Using reward model-based sampling is a fast and efficient way to adjust the system. For more complex or larger cases, a more advanced method can be used to process larger amounts of data. Both methods use the same trained predictor, keeping the process predictable. The system remains transparent and easy to manage. You can check the coefficients, and the search process is clear. The generator rules are still written and controlled by humans. The effort and reliability are not subject to machine control. (Shneiderman, 2020).

However, the system's ability to adapt depends on the quality and diversity of its training data. A system that learns from preferences is sensitive to evaluation patterns, and can be strict, too flexible, or "not enough" depending on the data processing and evolution itself.

Once a reliable, easy-to-understand preference system has been trained and tested by different people, the system can begin using more organized search methods. Simple methods like selecting items that meet certain criteria, ranking, or choosing from a variety of options can be used right away. As the dataset grows and institutional deployment scales, better and faster search methods can be added incrementally without changing the entire setup.

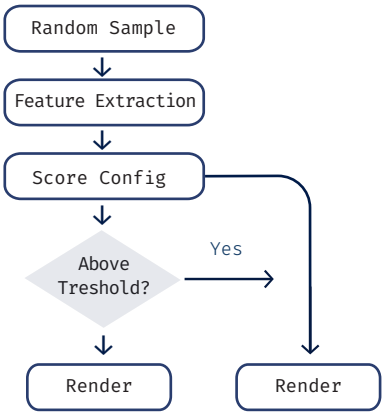


Fig. 6.2 Reward-Guided Generation

6.3 System Scalability and Institutional Use

If the sampling methods and preference modeling integrate to the system, it should remain useful over time; it must do more than adjust its settings. It should also advance in technology, organization, and its interactions with different institutions. Scalability here is not just about speed. It means the system can handle more feedback, changing branding rules, new settings, and various organizational uses, all without needing a complete rebuild.

The generator was designed to be assembled from separate parts. It has different layers: a drawing engine that always gives the same result, a way to suggest settings (which can be random or guided), a layer for saving and checking data, and, when needed, a model that learns what people like. This setup lets each part get better on its own. For example, new shapes can be added to the drawing part without changing the model that learns preferences. The reward model can be updated as new feedback comes in, without changing the drawing rules. Rules from organizations can be applied when selecting samples without disrupting data analysis.

This modular design is important for institutions, where systems need to be easy to maintain and adaptable as needs change.

As more departments participate, the dataset grows larger and more varied. Having more samples makes the results more reliable, while having different people rate things brings in a wider range of opinions, which can sometimes conflict. Scalability here also means being able to handle input from many departments. The system can be set up in different ways. It can use all the ratings together to find what most people in the organization like, or it can keep separate models for each department to match their own ways of judging.

In more advanced setups, it can create options that most people will like while also reducing disagreements between different raters. If the model is trained during evaluation rather than after a complete dataset, which can take a long time, the preference model should be updated periodically. Scheduled retraining cycles preserve stability, enable version control, and document the model's evolution over time.

From a technical standpoint, the chosen modeling method is easy to deploy. Logistic regression uses a simple calculation followed by a sigmoid function, which runs efficiently on regular computers or even in a web browser. There is no need for special machine learning hardware. This means the system can be used in schools and offices without relying on outside servers or GPUs (Smilkov et al., 2019). Even if the dataset grows to tens of thousands of samples, training is still manageable. Only if the number of parameters grows a lot would more advanced methods be needed. Besides technical issues, scalability also means making the system easy to use. Right now, the prototype uses a technical interface (GUI) that shows detailed settings. This works for research, but might be too complicated and "sterile" for wider use in organizations. It's possible to build a simpler, more organized interface while keeping all the current settings. They could be renamed to be more meaningful across departments, such as "structural density," "formality," "symmetry strength," or "expressive variation." These bigger controls would handle detailed settings in the background, so users do not have to deal with technical details. Also, simpler sliders, a different layout, and separating options after picking a shape would make it easier to use and speed up the design process.

```
User Interface
Layer
    ↓
Sampling Layer
(Random /
Smart Random /
Reward-Guided)
    ↓
Preference
Model (Optional
/ Updatable)
    ↓
Data Collection
& Retraining
Layer
    ↓
Deterministic
Drawing Engine
```

Switching from detailed settings to bigger ideas would make the system much easier to use. It would also better match how organizations work, since users usually want results that fit their communication goals, not just to try out different shapes. Even if a full redesign is not possible right now, it is important to keep this goal in mind. Scalability is not just about handling more data, but also about making the system easier for non-experts to use.

6.3 Ethical Considerations

Adding adaptive features to a generative system needs an ethical review of both what the system does and what it chooses not to do. In a place like Politecnico di Milano, new technology must follow clear rules. The system in this thesis is not an open, self-changing AI, but a controlled engine with only a few carefully limited adaptive features.

A key ethical choice in this project was to avoid the use of unclear or black-box AI systems. No outside connections, cloud-based models, or training tools are needed to use the system. The preference model is simple, easy to understand, and runs on local computers. The rendering engine always gives the same results for the same input.

Ethical responsibility is about more than just whether data is public or private. As Fiesler argues, thinking that data is 'already public' hides the possible harms of making guesses, spreading, or changing the meaning of that data. When used in an institution, the system must be easy to understand (Fiesler, 2019). A simple model that works in a clear, straightforward way meets these needs better than complex AI models that are hard to understand.

The system, therefore, remains intentionally limited and does not invent new visual brand logics because it works strictly within the defined rules of branding. Again, if the system focuses on the random or generative part, it introduces structural risk. It can gradually deviate from the established visual identity despite refinement, leading to iterative adaptation. While exploratory variation is valuable in design research, institutional branding requires coherence and stability.

For this reason, the adaptive features described here are added to a fixed set of design rules based on the institution's visual style. The reward model can affect choices within these rules, but cannot change them. Things like shape types, layout rules, and size limits stay within set boundaries. This control ensures that changes help keep things consistent rather than make them less so. The system cannot produce results that break the main brand rules because those rules are built into its operation.

This rule-based design emphasizes control and oversight in computer creativity. As Boden describes, creative exploration happens within a space defined by rules. In this system, that space is set by the institution. The system can move within that space, but cannot make it bigger without a person choosing to do so. Data stays within the institution because settings are configured locally, feedback is collected in controlled settings, and training occurs offline. This avoids the ethical ambiguities associated with public data reuse, as highlighted in discussions of dataset ethics and recontextualization. Even when data is technically accessible, repurposing it for algorithmic inference may violate contextual expectations.

By limiting the system to internally generated feedback and only using feedback given willingly within the institution, the project lowers the risk of problems with privacy, consent, and data sharing. Ethical risks remain. One is aesthetic narrowing. If the reward model overemphasizes certain configurations, the generator may converge toward repetitive patterns. This is not a privacy issue but a structural bias issue.

Adaptive features must stay clear and easy to change. Rules for variety and regular retraining help prevent the system from settling too soon on a single pattern.

The system should never be seen as the only judge of design quality. Its results show only trends within a small group and for a limited time. People within the institution still need to check its work. Institutional deployment requires documentation and version control. Each trained model should be archived with dataset version, training date, validation metrics and parameter configuration. These practices help ensure people are responsible and that results can be repeated. If the design changes, the model's changes can be checked. If bias is found, retraining can be monitored. Importantly, designers can always take control of the system. They can skip adaptive choices and return to random exploration.

```
Institutional
Rules (Outer
Boundary)
    ↓
Parametric
Space
    ↓
Adaptive Layer
(Reward Model)
    ↓
Human Oversight
```

Using adaptation is a choice, not a requirement. The ethical choices in this system are not just about data storage, as most of it is published. It is necessary to maintain the branding's autonomy and uniqueness and adapt the system so that those who are not specifically designers can use it without compromising the

branding's presentation. In addition, this should be an adaptive design for the institution, not its automation and research without limits. In this system, computer assistance strengthens the institution's identity rather than weakening it.

Conclusion

This research aimed to determine how a visual identity system can be structured and computerized without losing its unity, sense of ownership, or the institution's values. The project asked if branding, usually seen as a strict set of rules, could become flexible while still being managed.

The shape generator shows that an institution's visual style can be turned into a set of changeable rules. Instead of using fixed templates, the system uses shapes, sizes, and structure rules in a computer program. Each result can be traced back to clear settings, making the process repeatable, easy to check, and consistent with the main identity ideas.

In this system, randomness was used not to make things unpredictable, but to help try out different design options in an organized way. Testing the random segment showed that there were many possible designs, but few matched the institutional one, or at least the opinions of a small number of evaluators. This showed that the generator is not just a neutral tool but a place where people's tastes can differ.

The evaluation phase transformed personal and professional opinions into data that could be studied. In doing so, the project combined design testing with careful analysis. This step provided insight into many of the limitations of the existing system, as well as the fact that, without much randomization, it works well simply by organizing the parameter structure differently. The use of guided choice and preference learning did not change the basic visual rules. The designer's role is not diminished, as the system always relies on a geometric structure defined by branding.

It is important to note that this whole idea of future learning

implementation is not based on the use of artificial intelligence. Clear models, careful updates, and secure data storage can make the system responsive without losing control. This work proposes a careful and responsible approach to using generative systems in schools and universities. Scalability here means making the system long-lasting, not just faster. The design allows for slow and steady growth, like adding new reviewers, settings, or better ways to test, without making the system unstable. Keeping the steps of showing results, testing, checking, and modeling separate helps keep things organized and under control.

From an ethical standpoint, the system is designed with clear limits. By not relying on external data or complex systems, it remains transparent and follows human-centered AI principles. The generator does not change the institution's identity on its own. It works within set boundaries, allowing exploration while maintaining consistency.

In the end, this research offers a way to see branding systems as both dynamic and well-governed. It shows that parametric design, data-driven evaluation, and adaptive modeling can work together in a system that is clear and responsible to the institution. The shape generator is not meant to be a final design tool, but rather a method. It shows that institutional visual identity can be formalized, explored, tested, and improved using structured computational systems, without losing authorship or accountability.

In doing so, the project presents generative branding not as automatic design but as careful improvement. The structure is still created by people, adaptation is clear, and the institution's identity stays at the center.

a

Ahmad, M. (2022, January 26). An algorithm for polygons with rounded corners. Gorilla Sun. https://www.gorillasun.de/blog/an-algorithm-for-polygons-with-rounded-corners/

Amershi, S., Cakmak, M., Knox, W. B., & Kulesza, T. (2014). Power to the people: The role of humans in interactive machine learning. AI Magazine, 35(4), 105–120. https://doi.org/10.1609/aimag.v35i4.2513

Amershi, S., Weld, D., Vorvoreanu, M., Fourney, A., Nushi, B., Collisson, P., Suh, J., Iqbal, S., Bennett, P. N., Inkpen, K., Teevan, J., Kikin-Gil, R., & Horvitz, E. (2019). Guidelines for human-AI interaction. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (pp. 1–13). https://doi.org/10.1145/3290605.3300233

Another Graphic. (n.d.). Experimental typographic poster by Bureau Borsche. Retrieved February 22, 2026, from https://anothergraphic.org/

b

Bai, X., Guan, J., & Wang, H. (2019). A model-based reinforcement learning with adversarial training for online recommendation. In Advances in Neural Information Processing Systems, 32.

Boden, M. A. (1998). Creativity and artificial intelligence. Artificial Intelligence, 103(1–2), 347–356.

Bannerbear. (n.d.). Rule-based layout system. Retrieved February 21, 2026, from https://www.bannerbear.com/

c

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., & Amodei, D. (2017). Deep reinforcement learning from human preferences. In Advances in Neural Information Processing Systems, 30.

Clarke, M., & Joffe, M. (2025). Beyond replacement or augmentation: How creative workers reconfigure division of labor with generative AI (arXiv:2505.18938). arXiv. https://doi.org/10.48550/arXiv.2505.18938

Cross, N. (2018). Expertise in professional design. In K. A. Ericsson et al. (Eds.), The Cambridge handbook of expertise and expert performance (2nd ed., pp. 372–388). Cambridge University Press. https://doi.org/10.1017/9781316480748.021

d

Data Arts Team. (2011). dat.GUI [Computer software]. GitHub. https://github.com/dataarts/dat.gui

Douglas, D. M., Howard, D., & Lacey, J. (2021). Moral responsibility for computationally designed products. AI & Ethics, 1(3), 287–301. https://doi.org/10.1007/s43681-020-00034-z

Dunne, A., & Raby, F. (2013). Speculative everything: Design, fiction, and social dreaming. MIT Press.

Diversions Magazine. (2021). Chaumont Biennale 2021 identity system by DIA. Retrieved February 22, 2026, from http://www.diversions-magazine.com/

f

Fiesler, C. (2019). Ethical considerations for research involving (speculative) public data. Proceedings of the ACM on Human-Computer Interaction, 3(GROUP), 1–13. https://doi.org/10.1145/3370271

Frayling, C. (1993). Research in art and design. Royal College of Art Research Papers, 1(1).

Frontify. (n.d.). Brand automation guide. Retrieved February 21, 2026, from https://www.frontify.com/

Framer. (n.d.). Component-driven interface and rule-based partial automation. Retrieved February 21, 2026, from https://www.framer.com/

FIELD.IO. (2024). Instagram global gradient toolkit. Retrieved February 22, 2026, from https://www.field.io/

g

Gaver, W. (2012). What should we expect from research through design? In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 937–946). https://doi.org/10.1145/2207676.2208538

Gerstner, K. (1964/2007). Designing programmes. Lars Müller Publishers.

Gillies, M., Fiebrink, R., Tanaka, A., et al. (2016). Human-centred machine learning. In Proceedings of the 2016 CHI Conference Extended Abstracts (pp. 3558–3565). https://doi.org/10.1145/2851581.2856492

Gregersen, M. K., & Johansen, T. S. (2022). Organizational-level visual identity: An integrative literature review. Corporate Communications: An International Journal, 27(3), 441–456. https://doi.org/10.1108/CCIJ-06-2021-0068

Gunagama, M. G. (2018). Generative algorithms in alternative design exploration. SHS Web of Conferences, 41, 05003. https://doi.org/10.1051/shsconf/20184105003

Johnson, M. (2016). Branding: In five and a half steps. Thames & Hudson.

k

Kantosalo, A., & Riihiaho, S. (2019). Experience evaluations for human–computer co-creative processes. Connection Science, 31(1), 60–81. https://doi.org/10.1080/09540091.2018.1432566

Khosravi, H., Das, S., Al-Mamun, A., & Ahmed, I. (2024). Binary Gaussian copula synthesis (arXiv:2403.00965). arXiv. https://doi.org/10.48550/arXiv.2403.00965

Knaflic, C. (2019). Storytelling with data: Let's practice! Wiley.

Kretzschmar, M., Dammann, M. P., Schwoch, S., Braun, F., & Saske, B. (2024). Evaluating the current role of generative AI in engineering development and design. Procedia CIRP, 122, 829–834.

l

Liao, X., & Hu, X. (2023). Automatic layout algorithm for graphic language in visual communication design. International Journal of Advanced Computer Science and Applications, 14(8). https://doi.org/10.14569/IJACSA.2023.0140878

Light, A., Powell, A., & Shklovski, I. (2017). Design for existential crisis in the Anthropocene age. In Proceedings of the 8th International Conference on Communities and Technologies (pp. 270–279). https://doi.org/10.1145/3083671.3083688

Lyons, J. B., Sycara, K., Lewis, M., & Capiola, A. (2021). Human–autonomy teaming. Frontiers in Psychology, 12, 589585. https://doi.org/10.3389/fpsyg.2021.589585

m

Manovich, L. (2015). Data science and digital art history. International Journal for Digital Art History, 1, 14–34. https://doi.org/10.11588/dah.2015.1.21631

Manovich, L. (2020). Cultural analytics. MIT Press.

Manzini, E. (2015). Design, when everybody designs. MIT Press.

Marocco, S., Talamo, A., & Quintiliani, F. (2024). From service design thinking to activity theory. Frontiers in Artificial Intelligence, 7, 1303691. https://doi.org/10.3389/frai.2024.1303691

Matos, L. M., Azevedo, J., Matta, A., Pilastri, A., Cortez, P., & Mendes, R. (2022). CANE: A python module for categorical to numeric data preprocessing. Software Impacts, 13, 100359. https://doi.org/10.1016/j.simpa.2022.100359

McCormack, J., Dorin, A., & Innocent, T. (2004). Generative design: A paradigm for design research. In Proceedings of Futureground 2004.

Mechanic. (n.d.). Mechanic documentation. https://mechanic.design

Moline, K., & Hall, P. (Eds.). (2015). Experimental thinking / Design practices. Griffith University.

Mosqueira-Rey, E., Hernández-Pereira, E., Alonso-Ríos, D., Bobes-Bascarán, J., & Fernández-Leal, Á. (2023). Human-in-the-loop machine learning: A state of the art. Artificial Intelligence Review, 56(4),

3005–3054. https://doi.org/10.1007/s10462-022-10246-w

Mu, T., Helyar, A., Heidecke, J., et al. (2024). Rule-based rewards for language model safety (arXiv:2411.01111). arXiv. https://doi.org/10.48550/arXiv.2411.01111

Munro, R. (2021). Human-in-the-loop machine learning. Manning Publications.

p

Parikh, K., Kaufman, D. M., Levin, D. I. W., & Jacobson, A. (2025). Differentiable variable fonts (arXiv:2510.07638). arXiv. https://doi.org/10.48550/arXiv.2510.07638

Peckham, O., Raines, J., Bulsink, E., et al. (2025). Artificial intelligence in generative design. Designs, 9(4), 79. https://doi.org/10.3390/designs9040079

Politecnico di Milano. (2023). Digital brand guidelines – UI annex.

Politecnico di Milano. (2023). Politecnico di Milano brand manual. https://www.polimi.it/en/the-politecnico/communication/brand-identity-manual

Processing Foundation. (n.d.). Processing visualization example. Retrieved February 22, 2026, from https://processing.org/

p5.js contributors. (n.d.). p5.js generative system example. Retrieved February 22, 2026, from https://p5js.org/

Prototypo. (n.d.). Parametric typography interface documentation. Retrieved February 22, 2026, from https://www.prototypo.io/

r

Reas, C., McWilliams, C., & Barendse, J. (2011). Form+code in design, art, and architecture. Princeton Architectural Press.

S

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. IBM Journal of Research and Development, 3(3), 210–229.

Saadi, J., & Yang, M. (2023). Observations on generative design tools. Proceedings of the Design Society, 3, 2805–2814. https://doi.org/10.1017/pds.2023.281

Schön, D. A. (1983). The reflective practitioner. Basic Books.

Shneiderman, B. (2020). Human-centered artificial intelligence. International Journal of Human–Computer Interaction, 36(6), 495–504.

Sharp, H., Preece, J., & Rogers, Y. (2019). Interaction design. John Wiley & Sons.

Smilkov, D., Thorat, N., Assogba, Y., et al. (2019). TensorFlow.js: Machine learning for the web and beyond (arXiv:1901.05350). arXiv.

Spelova, A. (2024, August 13). What is the difference between visual identity, brand identity and branding? Rocketscience.

t

Tierney, K. D., Karpen, I. O., & Westberg, K. (2022). Brand meaning and institutional work. Journal of Business Research, 151, 244–256.

Trautmann, L., & Piros, A. (2020). The concept of EmPatGen. SN Applied Sciences, 2(5), 982.

V

Victor, B. (2010). Magic ink: Information software and the graphical interface.

W

Wang, P., Peng, D., Li, L., et al. (2019). Human-in-the-loop design with machine learning. In Proceedings of the Design Society (pp. 2577–2586).

Wheeler, A. (2009). Designing brand identity (3rd ed.). John Wiley & Sons.

Whitelaw, M. (2012). Towards generous interfaces for archival collections. Comma, 2012(2), 123–132.

Wiestner, D., Spiller, R., Bouchedoub, A., & Hasan, M. (2024). Automation of brand management. In Communication and Applied Technologies (pp. 127–138).

y

Yıldırım, E. (2024). Towards intelligent architecture. IZLIK: Journal of Architecture and Design, 1(2).

z

Zimmerman, J., Forlizzi, J., & Evenson, S. (2007). Research through design. In Proceedings of the SIG-CHI Conference (pp. 493–502).