
Travail de fin d'études / Projet de fin d'études : Impact of context familiarity on computational design logic appropriation

Auteur : Garnavault, Xavier

Promoteur(s) : 451; Leclercq, Pierre

Faculté : Faculté des Sciences appliquées

Diplôme : Master en ingénieur civil architecte, à finalité spécialisée en ingénierie architecturale et urbaine

Année académique : 2021-2022

URI/URL : <http://hdl.handle.net/2268.2/16273>

Avertissement à l'attention des usagers :

Tous les documents placés en accès ouvert sur le site le site MatheO sont protégés par le droit d'auteur. Conformément aux principes énoncés par la "Budapest Open Access Initiative"(BOAI, 2002), l'utilisateur du site peut lire, télécharger, copier, transmettre, imprimer, chercher ou faire un lien vers le texte intégral de ces documents, les disséquer pour les indexer, s'en servir de données pour un logiciel, ou s'en servir à toute autre fin légale (ou prévue par la réglementation relative au droit d'auteur). Toute utilisation du document à des fins commerciales est strictement interdite.

Par ailleurs, l'utilisateur s'engage à respecter les droits moraux de l'auteur, principalement le droit à l'intégrité de l'oeuvre et le droit de paternité et ce dans toute utilisation que l'utilisateur entreprend. Ainsi, à titre d'exemple, lorsqu'il reproduira un document par extrait ou dans son intégralité, l'utilisateur citera de manière complète les sources telles que mentionnées ci-dessus. Toute utilisation non explicitement autorisée ci-avant (telle que par exemple, la modification du document ou son résumé) nécessite l'autorisation préalable et expresse des auteurs ou de leurs ayants droit.



University of Liège
Faculty of Applied Sciences

Impact of Context Familiarity on Computational Design Logic Appropriation

Master's thesis in order to a master's degree in
Architectural Civil Engineering, by

Xavier GARNAVAULT

Supervisors: Prof. Aurélie DE BOISSIEU
Prof. Pierre LECLERCQ

Jury: Mohamed-Anis GALLAS
Xaviera CALIXTE

Table of Contents

List of figures.....	5
1. Introduction.....	7
2. Computational design in AEC.....	8
2.1. Historical context of computation in AEC.....	8
2.2. Defining Computational design.....	9
2.2.1. Parametric design.....	9
2.2.2. Generative design.....	9
2.2.3. Algorithmic design.....	10
2.3. CD toolsets.....	11
2.3.1. Grasshopper.....	11
2.3.2. Dynamo.....	11
2.3.3. Community solutions.....	12
2.4. Main problems and challenges.....	13
2.5. Current Development direction.....	14
2.5.1. Computational design platforms.....	14
2.5.2. Interoperability and open source.....	16
2.6. Culture shift.....	17
3. Research Question.....	18
4. Research Methodology.....	19
4.1. Design logic implementations.....	19
4.1.1. Origin.....	20
4.1.2. Algorithm overview.....	21
4.1.3. Implementations.....	22
4.2. User selection.....	25
4.3. Experiment Protocol.....	25
4.3.1. Experiment environment variables.....	25
4.3.2. Subject Characterization.....	26
4.3.3. Tool interactions.....	27
4.3.4. Subject feedback.....	27
4.3.5. Protocol validation.....	27
4.4. Data harvesting.....	28
4.5. Data Cleaning.....	29
4.5.1. Survey data cleaning.....	29
4.5.2. Tool interactions data cleaning.....	30
4.6. Data Engineering.....	30

4.6.1.	Iterations data engineering.....	32
4.6.2.	Modification phases.....	33
4.6.3.	Unique values.....	34
4.7.	Data visualization.....	35
4.7.1.	feedback.....	36
4.7.2.	global_analysis.....	37
5.	Data analysis.....	40
5.1.	Harvested data.....	41
5.1.1.	Successful experimentations and collected data points.....	41
5.1.2.	Characterization of the population.....	41
5.2.	Differences between different Contexts of interaction.....	42
5.2.1.	Survey.....	42
5.2.2.	Number of iterations.....	43
5.2.3.	Number of modification phases.....	43
5.2.4.	Unique Values.....	44
5.2.5.	Summary.....	44
5.3.	Order of interaction impacted behavior.....	45
5.3.1.	Survey.....	45
5.3.2.	Number of iterations.....	46
5.3.3.	Number of Modification Phases.....	46
5.3.4.	Number of unique values.....	47
5.3.5.	Summary.....	47
5.4.	Previous software experience impacted the interaction.....	48
5.4.1.	AutoCAD.....	48
5.4.2.	SketchUp.....	49
5.4.3.	Rhino.....	50
5.4.4.	Grasshopper.....	51
5.4.5.	Archicad.....	52
5.4.6.	Revit.....	53
5.4.7.	BIM.....	54
5.4.8.	Modeling.....	55
5.5.	Context familiarity impacted behavior.....	56
5.5.1.	Grasshopper.....	56
5.5.2.	Hybrid.....	60
5.5.3.	Rhino.....	64
6.	Results and discussions.....	68

7. Conclusion.....	69
8. References	70
9. Appendix	72
Appendix 1 : Average by AutoCAD	72
Appendix 2 : Average by SketchUp	73
Appendix 3 : Average by Rhino	74
Appendix 4 : Average by Grasshopper	75
Appendix 5 : Average by Archicad.....	76
Appendix 6 : Average by Revit	77
Appendix 7 : Average by BIM	78
Appendix 8 : Average by Modeling	79
Appendix 9 : Average by Order	80
Appendix 10 : Average by Context.....	81
Appendix 11 : Grasshopper by AutoCAD	82
Appendix 12 : Grasshopper by SketchUp.....	83
Appendix 13 : Grasshopper by Rhino.....	84
Appendix 14 : Grasshopper by Grasshopper	85
Appendix 15 : Grasshopper by Archicad	86
Appendix 16 : Grasshopper by Revit	87
Appendix 17 : Grasshopper by BIM.....	88
Appendix 18 : Grasshopper by Modeling.....	89
Appendix 19 : Grasshopper by Order.....	90
Appendix 20 : Hybrid by AutoCAD	91
Appendix 21 : Hybrid by SketchUp	92
Appendix 22 : Hybrid by Rhino.....	93
Appendix 23 : Hybrid by Grasshopper	94
Appendix 24 : Hybrid by Archicad	95
Appendix 25 : Hybrid by Revit.....	96
Appendix 26 : Hybrid by BIM.....	97
Appendix 27 : Hybrid by Modeling.....	98
Appendix 28 : Hybrid by Order.....	99
Appendix 29 : Rhino by AutoCAD	100
Appendix 30 : Rhino by SketchUp	101
Appendix 31 : Rhino by Rhino	102
Appendix 32 : Rhino by Grasshopper.....	103
Appendix 33 : Rhino by Archicad	104

Appendix 34 : Rhino by Revit 105
Appendix 35 : Rhino by BIM 106
Appendix 36 : Rhino by Modeling 107
Appendix 37 : Rhino by Order 108

List of figures

Figure 2.1-1 : CAD to Computational Design. Source: (de Boissieu, Introduction to Computational Design: Subsets, Challenges in Practice and Emerging Roles, 2022)	8
Figure 2.2-1 : Venn diagram of Parametric Design (PD), Generative Design (GD), and Algorithmic Design (AD). (Caetano, Santos, & Leitão, 2020)	9
Figure 2.2-2 : Waterloo truss system parametric logic (credits: Shane Burger).....	9
Figure 2.5-1 : Hypar UI example	14
Figure 2.5-2 : Testfit UI example.....	15
Figure 4.1-1 : Output example	20
Figure 4.1-2 : Step 1	21
Figure 4.1-3 : Step 2	21
Figure 4.1-4 : Step 3	21
Figure 4.1-5 : Final result	21
Figure 4.1-6 : Grasshopper implementation view	22
Figure 4.1-7 : Hybrid implementation view	23
Figure 4.1-8 : Result preview	24
Figure 4.1-9 : Principal Menu	24
Figure 4.1-10 : Grid submenu	24
Figure 4.1-11 : Opening submenu.....	24
Figure 4.3-1 : Software proficiency survey	26
Figure 4.3-2 : Experiment sequence	27
Figure 4.4-1 : Data harvesting example	28
Figure 4.5-1 : Feedback DataFrame example (1 st interaction survey).....	30
Figure 4.6-1 : iterations_df example.....	32
Figure 4.6-2 : diff_df example	32
Figure 4.6-3 : imp_df example	32
Figure 4.6-4 : diff_df example	33
Figure 4.6-5 : diff_diff_df example	33
Figure 4.6-6 : phase_df example.....	33
Figure 4.6-7 : imp_df.....	34
Figure 4.6-8 : unique_df.....	34
Figure 4.6-9 : Main DataFrame example	34
Figure 4.7-1 : feedback graph example.....	36
Figure 4.7-2 : global_analysis (iterations_df, time_iterations_df, ['Time','Rate','Total','CV'], 'Iterations', 'Level', None, 'Grasshopper', False).....	37
Figure 4.7-3 : global_analysis (iterations_df, time_iterations_df, ['Time','Rate','Total','CV'], 'Iterations', 'Level', None, 'Grasshopper', True).....	38
Figure 4.7-4 : Example of significant value graphic analysis.....	38
Figure 4.7-5 : Rate evolution.....	39
Figure 4.7-6 : Parameter values.....	39
Figure 5.1-1 : Modeling proficiency distribution	41
Figure 5.1-2 : CD proficiency distribution	41
Figure 5.1-3 : BIM proficiency distribution	41
Figure 5.1-4 : Software proficiency chart.....	41
Figure 5.2-1 : Survey data by context of interaction	42
Figure 5.2-2 : Iterations by Context	43
Figure 5.2-3 : Modification phases by Context	43
Figure 5.2-4 : Unique values by Context	44
Figure 5.3-1 : Survey data by Order	45
Figure 5.3-2 : Iterations by Order of interaction.....	46
Figure 5.3-3 : Modification phases by Order of interaction	46

Figure 5.5-1: Grasshopper survey data by Grasshopper level	56
Figure 5.5-2 : Detailed study of the number of iterations in Grasshopper, by Grasshopper level.....	57
Figure 5.5-3 : Detailed study of the number of modification phases in Grasshopper, by Grasshopper level.....	58
Figure 5.5-4 : Detailed study of the number of unique values in Grasshopper, by Grasshopper level	59
Figure 5.5-5 : Hybrid survey data by average Rhino-GH level	60
Figure 5.5-6 : Detailed study of the number of iterations in the Hybrid context, by average Rhino and Grasshopper level	61
Figure 5.5-7 : Detailed study of the number of modification phases in the Hybrid context, by average Rhino and Grasshopper level	62
Figure 5.5-8 : Detailed study of the number of unique values in the Hybrid context, by average Rhino and Grasshopper level	63
Figure 5.5-9 : Rhino survey data by Rhino level.....	64
Figure 5.5-10 : Detailed study of the number of iterations in Rhino, by Rhino level	65
Figure 5.5-11 : Detailed study of the number of modification phases in Rhino, by Rhino level	66
Figure 5.5-12 : Detailed study of the number of unique values in Rhino, by Rhino level	67

1. Introduction

I first discovered computational design in 2018, through an introductory course to 3d modeling which contained a module on Parametric design in Rhino and Grasshopper. Though I did not realize and appreciate it at the time (using only SketchUp for the rest of my project), I can now say without a doubt that that introduction had a deeper impact on my life and my future than any other course during my studies.

After that introductory course, I spent nearly a year without using grasshopper, until I purchased a 3d printer with a clear vision for generating complex geometry to be printed. Through this exercise and objective, I rediscovered grasshopper and realized how powerful a tool it could be. This naïve exploration of grasshopper gave me the brash confidence to use it for the 4th year design studio project, in which I developed a fully parametric model of the complex geometrical building my team and I designed, which allowed us not only to control and generate this complex geometry but also iterate designs faster than would have been possible in even simple buildings using classical tools. This was my first practical experience of the new ways of working that computational design allowed and served as a revelation: not only did I learn a great deal during the project, but I also discovered all that was still left to learn.

Since then I have devoted myself to learning as much as possible, but all the while I could not help but wonder why my classmates who had been exposed to the same introductory course as I had did not also attempt to use and develop these tools and . Typical answers they gave were lack of time to learn these tools or their preference for tools with which they were already familiar.

Later on, during my internship or working as a student in a small architectural practice, I realized how inefficient standard workflows were, and the possibility for integrating even simple computational design to automate or enrich workflows.

These experiences, along with my deep interest for computational design, are what drove me to conduct this work on Computational design logic appropriation, by focusing on whether lack of familiarity in a software environment is really an obstacle to using computational design within it.

To conduct this work, we will first provide an overview of computational design within the AEC industry in section 2 that will help the framing of the research question in section 3.

In section 4, we will present the research methodology that was developed to then proceed to the data analysis in section 5.

Finally sections 6 and 7 will be reserved for the discussion of the results, a retrospective look on the implemented methodology, avenues of further development of this research and a final summary that will conclude this work.

2.2. Defining Computational design

As previously mentioned, while there is no definitive definition for Computational Design, we can distinguish three subsets: parametric, generative, and algorithmic design (Caetano, Santos, & Leitão, 2020). In the following sections, we will present an overview of these subsets, their main characteristics, as well as provide an example for each subset. The link and overlap of these subsets are represented in Figure 2.2-1.

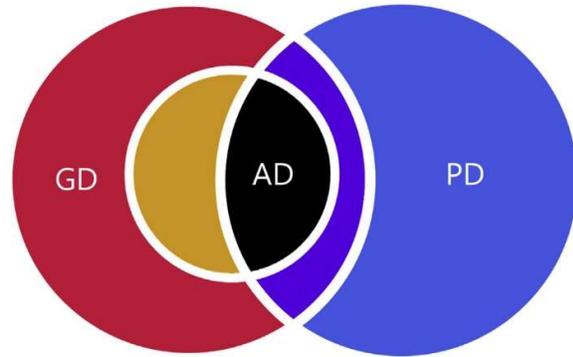


Figure 2.2-1 : Venn diagram of Parametric Design (PD), Generative Design (GD), and Algorithmic Design (AD). (Caetano, Santos, & Leitão, 2020)

2.2.1. Parametric design

The first subset is parametric design. Parametric design is composed of a system of clear inputs on which rules and constraints are applied to transform these inputs into the desired outputs. One specificity of parametric design is that this link is unidirectional: this, along with the more obvious relationship between input and output is the reason parametric design is more easily apprehended by beginners, particularly through the use of visual programming toolsets, as will be further explained in section 2.3. In parametric design, the designer can explore the design not only by operating on the parameters (inputs) but also by modifying the rules applied to them.

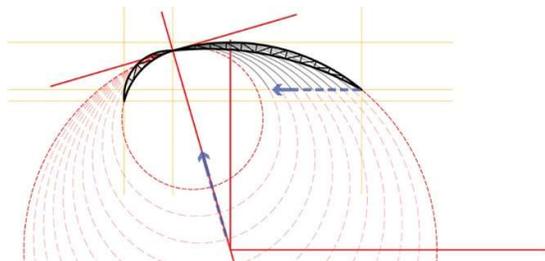


Figure 2.2-2 : Waterloo truss system parametric logic (credits: Shane Burger)

One example of the use of parametric design in AEC is Grimshaw's Waterloo train station. This project, designed in 1993, was constrained by a difficult site. A parametric design was implemented to describe the variable span of roof trusses in which the structural logic is consistent. This structural logic is composed in two parts: a constant arc section and a variable arc section which adapts to the constraints of the site (Figure 2.2-2)¹.

2.2.2. Generative design

A distinctive characteristic of generative design is its use of algorithms that are able to generate complex solutions from simple inputs (Van der Zee & De Vries, 2008). These algorithms function in such a way that the outputs are not predictable from the inputs, but rather satisfy certain criterion to form a solution-space. For this reason, generative design is often used to find "optimal" solutions for complex problems with undefined boundary conditions. Some common algorithms used for this are evolutionary algorithms, L-systems, cellular automata, or swarm systems.

This type of design is notably used to solve problems such automatic layout of units or furniture in web-based platforms that offer this as a service, as will be seen in section 2.5.1.

¹ Image source: <http://shaneburger.com/2011/08/designing-design/waterloo-geometrydiagram-cropped/>

2.2.3. Algorithmic design

Algorithmic design is defined more loosely and can sometimes overlap with parametric or generative design. It is more dependent on computational thinking and a direct traceability can be found between the generated results and the inputs. The main difference between algorithmic design and parametric design is that where in parametric design there is a direct link between input and output, it is acyclical by nature whereas in AD a logic can be applied continuously until a certain condition is met. Additionally, in AD the actions performed for each step can adapt to the inputs and make use of conditional logic; thus, while the result can always be traced back to the input and is consistent, it cannot always be easily predicted. Some examples of the use of algorithmic design are the Morpheus hotel by Zaha Hadid architects or projects in which form-finding plays a key role, such as in the Musmeci bridge.

2.3. CD toolsets

While the use of CD can find its origins in classical text-based programming which is still in use and indispensable in order to develop and apply more complex algorithms, this requires a skillset not typically found within the AEC industry. Because of that, the history of the use of computational design in AEC is intrinsically linked to the development of Visual programming, which is the most common way in which it is used and developed today.

Visual programming is a method of defining computational logic visually by connecting blocks of preestablished logic or parameters to one another, following a continuous flow of logic that is inherently acyclical, making it particularly adapted to the creation of parametric design which is its main use case. In the following sections we will focus first on the tools that implemented this paradigm and allowed the wider use of computational design, followed by how these tools were then extended to address some of their limitations or extend their functionality.

2.3.1. Grasshopper

Grasshopper was developed by David Rutten in 2007, then called “Explicit History”, as a complement to the existing history tool included in Rhino 4.0. Where the existing history tool kept track of the different steps taken by the user while modeling and the relation between different geometries, this new tool allowed the user to precisely define the different steps in logic leading to the desired outcome geometry. While this presents similarities with the existing Generative Components tool developed by Bentley, its implementation under the form of visual programming made it much more accessible to those without prior programming experience. One can simply connect and combine different blocks of operational logic to create the desired outcome. Although it was first released in 2007, the first stable release arrived in 2013, and has been included in Rhino by default since version 6. On April 1st, 2022, an alpha release of GH2 was made available to testers of Rhino’s WIP build.

2.3.2. Dynamo

Dynamo was first created by Ian Keough sometime before 2009², and open-sourced shortly afterwards. It enables users to interact with Revit’s API without explicit scripting to manipulate BIM objects and systems in Revit. Although it was created with the explicit aim to be used with Revit, in theory it could be used with another framework.

² <https://dynamobim.org/qa-about-dynamo/>

2.3.3. Community solutions

While tools such as grasshopper are already very capable “out of the box”, part of what allowed these tools (specifically grasshopper) to become so used and appreciated are the thriving communities around them, and the individuals who not only created solutions in the form of plugins to overcome some of grasshopper’s limitations and/or extend its capabilities, but also made these tools freely available to all.

HumanUI

HumanUI³ is a plugin that was developed by Andrew Heumann in 2015 (and open-sourced in 2016) to overcome a specific problem: Grasshopper’s intimidating UI for people unfamiliar with it. He states having been frustrated while working at NBBJ that while he developed scripts for others, for every change needed he had to be solicited or “babysit” a user. This plugin allows a designer to create a custom UI so that other users can interact with the underlying script without having to stay in grasshopper’s (or even Rhino’s) interface or design environment. One can create completely custom dashboards with friendly interfaces and with only the chosen parameters and outputs.

Kangaroo

Kangaroo is a live physics engine for grasshopper first developed by Daniel Piker in 2010 with this intention: *“The intention is that the various types of form-finding and feedback this allows could inform and enable some new ways of designing structures.”*⁴

A version 2 of kangaroo was released in 2015 and is included by default in grasshopper since Rhino 6.

Ladybug Tools

Ladybug tools is a set of tools aimed at environmental design. It was first developed starting in 2012 with the ladybug plugin by Mostapha Sadeghipour Roudsari, who states: *“I couldn’t stand the repetitive, simplified and disconnected workflows that I had to use on a daily basis as well as the overall lack of knowledge about environmental building design. I wanted educate more people about the principles of environmental building design and that happened to be through Ladybug!”*⁵

Ladybug as a plugin was first released in 2013, in the form of components for weather data visualization as well as solar radiation and sunlight analysis.

This initial plugin was followed by HoneyBee, which was released in 2014 with the aim to provide a connection between grasshopper and several validated daylighting and energy simulation engines.

Development continued and in 2016, Ladybug and Honeybee were rewritten in order to be used cross platform across Grasshopper and Dynamo.

In 2017, computational fluid design (CFD) was made available through the release of dragonfly which is based on OpenFOAM⁶.

³ <https://github.com/andrewheumann/humanui>

⁴ <https://www.grasshopper3d.com/profiles/blogs/project-kangaroo-live-3d>

⁵ <https://www.ladybug.tools/about.html>

⁶ <https://www.openfoam.com/>

2.4. Main problems and challenges

As we have already stated in the previous section, the early adoption of computational design in architecture was limited by the necessary computer science knowledge required to implement it. Even with the development and inclusion of tailored programming languages such as AutoLISP⁷ within CAD software environments, the use of computational design within architecture was extremely limited before the early to mid-2000s when visual programming tools appeared.

Although the arrival of these tools enabled easier access to computational design, usually specifically parametric design, this also came with its challenges. Where other “traditional” CAD tools and workflows were analogous to well established manual versions, parametric modelling necessitates another approach that designers were less familiar with. In order to construct a parametric model, one has to first determine which parameters are needed and how to link them together. This requires significant “Front Loading”, as described by Davis (2013). In his thesis, Daniel Davis states: *“This upfront planning can be challenging, particularly in a process as notoriously hard to anticipate as the design process.”*

Another challenge discussed by Davis is that in addition to having to figure out how a parametric model works upfront and what is needed to create it, in order to be able to use it effectively, a designer has to try to anticipate how it will be used and construct a flexible model. He summarizes it as such: *“(…), the skill of anticipating flexibility is getting the balance right between too much and too little flexibility.”* Indeed, if every step of the logic is made accessible for modification, the number of parameters a designer has to interact with becomes such that all the advantages of parametric modelling are lost compared to classic modelling. He also describes other challenges^{8,9,10} that he identifies as obstacles to the more widespread adoption and effective use of parametric design in architecture.

These challenges are also reported by other people, and served as the impetus for the creation of tools or plugins that aimed to solve these issues (such as HumanUI presented in section 0) and are part of the driving force for the current direction of development in this field as we will see in section 2.5.

Aside from the tool-based challenges, the novel approach that the use of these tools and development of custom CD logic necessitates, also called *Computational Thinking* (Menges & Ahlquist, 2011), is not part of the traditional architectural culture. This has led to evolutions in education and the emergence of new roles, as will be explained in section 2.6.

Finally, as identified by Stals (Stals, Elsen, & Jancart, Practical Trajectories of Parametric Tools in Small and Medium Architectural Firms, 2017), ignorance of the subject as well as perceived difficulty prevent many smaller and medium architectural practices from even attempting to integrate these practices in their work. This is particularly problematic, as practices at this scale have the most impact on the built environment and as such will have a significant role to play in addressing the challenges that face the industry. Furthermore, the scale and type of projects that these practices are traditionally responsible for, the “fat middle” (Davis, CAD’s Boring Future and Why it’s Exciting, 2021), will inevitably see the widespread use of computation, if not by architects, then by others (section 2.5.1).

⁷ AutoLISP is an integrated programming language for interacting with AutoCAD, first appearing in 1986.

⁸ Cases where the original logic can no longer be adapted or breaks

⁹ Unexpected, unwanted, and unseen changes due to the variation of a parameter

¹⁰ Obstacles designers of parametric models have when it comes to the reuse and sharing of their created models

2.5. Current Development direction

In reaction to the problems stated in section 2.4, and in addition to the natural typical development of all software environments, two development trends are of particular interest, even more so as they are not spearheaded by the traditional giants of the architectural software industry, in part due to the criticism these giants have faced in recent years (Davis, Architects versus Autodesk, 2020). These trends are:

- the development of web-based platforms (section 2.5.1 **Error! Reference source not found.**)
- the focus on interoperability and open-source development (section 2.5.2)

2.5.1. Computational design platforms

The past few years have seen the development of several cloud-based platforms that aim to enable the use of computational design by non-specialists. Of these platforms, we will discuss two that are already in advanced stages of development and indicative of the two directions that can be seen in such platforms.

The first platform, Hypar¹¹, is centered around the easy reuse of logic (computational or otherwise) facilitate the generation of building designs, rather than automate “drawing walls”, and not “start every building project from a blank page”¹². In this platform, users interact through a user-friendly UI (Figure 2.5-1) to combine pre-existing logic blocks from a library, with the option for specialists to create and integrate their own logic blocks (by easily porting existing logic developed in other environments such as Grasshopper, Dynamo or Excel to name a few) that they can then also make freely¹³ available to all, keep private or share with selected individuals. Once the desired combination of functions is constructed, users can easily explore different solutions, make changes, and share the logic through a simple URL for collaborators or even clients to interact with.

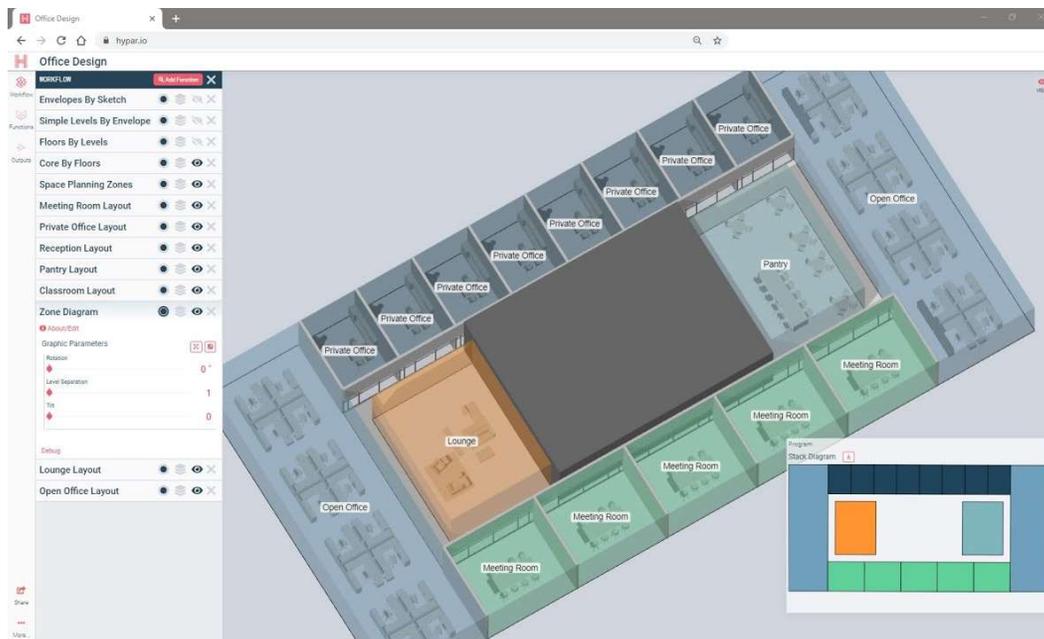


Figure 2.5-1 : Hypar UI example

¹¹ <https://hypar.io/>

¹² <https://hypar.io/about/story>

¹³ There are potential plans for a Marketplace in which designers could sell or licence the use of their functions, but this is not yet the case.

The second platform, Testfit¹⁴, aims to provide a service in the form of CD tools tailored for the study of feasibility of real estate developments. It uses complex generative design algorithms and the power of cloud computing to allow real estate developers, general contractors and architects to quickly generate massing, define unit types and automate their layout while data such interior area, building efficiency, unit cost, and others are generated to help inform design (Figure 2.5-2). This data and geometry can then be exported in various formats for the project to be further developed elsewhere.

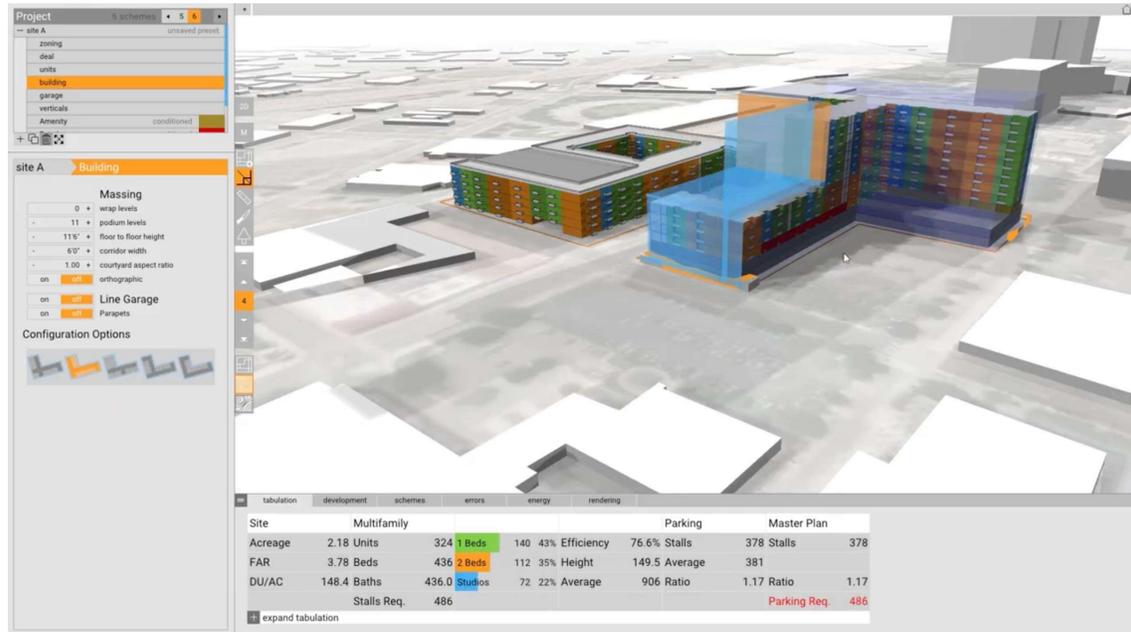


Figure 2.5-2 : Testfit UI example

¹⁴ <https://testfit.io/>

2.5.2. Interoperability and open source

The issue of interoperability between software (especially software from differing software providers) has long been a limiting factor for developing more complex workflows and collaboration. Finally, in this age of data, the subject of data propriety (Davis, 2020) (Fok & Picon, 2016) and data archival is ever more present.

Rather than continue in this direction, there seems to be a trend of developments that address these issues, notably through the use of open-source licensing and development.

One such example is the development of Speckle, an open-sourced cloud-based platform first developed by Dimitrie Stefanescu in 2015, with the explicit aim of addressing and handling *interoperability between software silos, real time collaboration, data management, versioning and automation*.¹⁵

Another example of opensource making its way into architecture and computational design is Blender¹⁶. Blender is a free open source 3d creation suite. While it is not currently widely used by the AEC industry, the release¹⁷ of the geometry nodes feature which brought¹⁸ visual programming (making it a viable option for computational design), and the development of projects such as BlenderBIM¹⁹ and Topologic²⁰ indicate that this may change.

Other projects exist such as IfcOpenShell²¹ and ifc.js²² which aim to enable individuals to develop their own platforms and tools.

¹⁵ <https://speckle.systems/about/>

¹⁶ <https://www.blender.org/>

¹⁷ As of Blender 2.92, released February 25, 2021

¹⁸ Although the Sverchock add-on already allowed a version of visual programming in Blender, the default inclusion and integration of geometry nodes brought it to the masses.

¹⁹ <https://blenderbim.org/>

²⁰ <https://topologic.app/>

²¹ <http://www.ifcopenshell.org/>

²² <https://ifcjs.github.io/info/>

2.6. Culture shift

In this section, we will investigate the perception of Computational Design in AEC, and how that perception is evolving through a culture shift and the emergence of new profiles and roles.

As we have already mentioned in previous sections, the adoption of computational design in AEC has not been as widespread as could have been expected given the advantages it promises. As such, several researchers have investigated this very topic. While some have identified elements such as the paradigms of new ways of thinking (Carpo, 2017) (Menges & Ahlquist, 2011) (Oxman, 2017), or challenges directly resulting from the tools used (Davis, 2013), one research angle that is of particular interest and directly linked to the research question of this study is the interrogation of the perception of computational design in “everyday” practitioners working in small and medium sized architecture practices, as done by Adeline Stals (Stals, 2019). During the course of their thesis, they conducted interviews and surveys in which they notably asked architects about their knowledge of parametric design, and what obstacles prevent them from using it. From the answers we can learn that while many architects are unfamiliar with parametric design or have trouble defining it, the main obstacles **perceived** are difficulties in learning and staying up to date.

While learning computational design goes beyond simply using a tool or software and entails additional effort and a different approach (Peters & De Kestelier, 2013), it is also becoming more widespread in education (Gallas, Jacquot, Jancart, & Delvaux, 2015) (Vrouwe, Dissaux, Jancart, & Stals, 2020) and as such every year the amount of architects with at least superficial knowledge of computational design increases.

From this pool of people who are receptive to the added value that computational design can have, there is an emergence of a new profile, that of the “super-user” (Deutsch, 2019). Characteristics of a super-user include a combination of soft-skills, strong technical expertise, and the ability to provide connections and provide structures that help those around them. In addition to using CD themselves, super-users can be the catalyst for the use and adoption of computational design practices by those around them (de Boissieu, 2020).

3. Research Question

From the previous section addressing the history and current state of computational design in AEC, it is clear that while the use of CD has been growing over the past few years, the industry as a whole is still largely ignorant or indifferent to this concept, even as they face ever growing pressure and challenges for which CD would be a precious resource...

At the same time, individuals and small initiatives are developing powerful tools and platforms that may very well disrupt the industry, either by the automation or power they employ through generative design and AI, or by adopting the practices of open-source software development and licensing that change the way we work.

Furthermore, the emergence of new profiles, so called super-users, and the adoption of Computational Design in education leading to a more informed and receptive culture, may finally allow CD to permeate through the industry.

However, even with these tools, profiles, and young architects which are receptive, there will still subsist a critical mass of current practitioners who are neither informed nor particularly receptive to these new ways of doing things. But in the face of the challenges the industry faces and will continue to face, it is unacceptable to simply wait for a natural turnover to provide this digital turn, even more so due to the rate of development in computing.

We must then find ways of rendering Computational Design accessible to them. Even if they do not create these design logics, non-specialist users can still benefit from them.

Given the common excuse of not knowing a certain program and the current trend for interoperability and web-based platforms, this work will attempt to provide a basis for the evaluation of the impact that familiarity with a given context of interaction has on a user's ability to understand and use it; and through the analysis of experiments, provide a set of initial results that may inform further research on the subject.

4. Research Methodology

To attempt to answer the research question, we will perform a quantitative analysis on the way a subject interacts with an unfamiliar computational design logic in a context and compare this analysis to the subject's experience in that and other software contexts. In order to source the data for this analysis, we will perform a study in the form of an experiment to provide a common computational design logic to interact with as well as a common design scenario to use it on.

The chosen subject pool will be described in section 4.2, the experiment protocol will be presented in section 4.3, the raw analysis data and the method for harvesting it is explained in section 4.4, the way this data will be processed and explored for analysis in sections 4.5 and 4.7.

In the following section (4.1), we will present the design logic that serves as the base for the experiments of this study, including its origin (section 0), a high-level overview of its inner workings (section 0) and the 3 implementations developed from it (section **Error! Reference source not found.**).

4.1. Design logic implementations

To serve as a base logic to be manipulated for the study, it was chosen to reuse an existing design situation as well as an existing script developed by me. My in-depth knowledge of the developed design logic allowed the identification of the various issues and points of interest, while the experimentation protocol allowed to anticipate possible biases. It provided a ready-made base script that presented sufficient complexity at distinct levels (incorporating parametric, generative, and algorithmic design). This enabled us to form conclusions from the study that could be extended throughout the computational design landscape, which would not have been the case if the script had been solely parametric or purely based on workflow automation.

4.1.1. Origin

The script that served as a base for the study was initially developed by myself in the spring of 2020 for a design project course during my Erasmus exchange in Barcelona. This project took the form of an assembly of prefabricated modular units, which led to the CD strategy to compose the façades for each of the units.

In the spirit of the adaptability that was sought and provided by the structural system (a principal timber framing system filled by secondary lightweight wood framed panels filled with insulating hemp blocks), and rather than simply copy the façades or constrain them to alignments, I developed a computational design to confer a unique identity to each unit while remaining in the confines of the grid system.

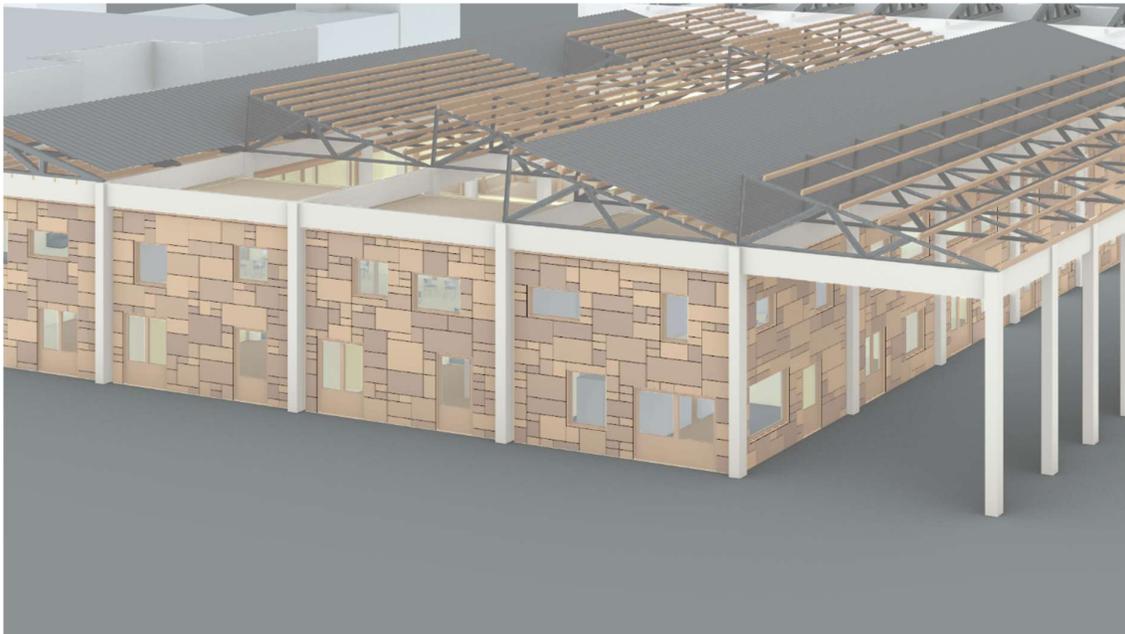


Figure 4.1-1 : Output example

The aim was to be able to tailor the positioning and size of every window and door of every unit based on desired metrics, such as views, daylight, or solar gains, and considering the local context of each unit. This positioning and sizing would fall within a grid linked with the structural framing and the size of the smallest sized ceramic tiles which would then form the cladding. An example of a solution for the project can be seen in Figure 4.1-1.

4.1.2. Algorithm overview

Given the repetitive nature of this modular and unit-based design, types are easily assigned to each floor of each unit, and for each type established a series of constraints (such as average size and variability) that would determine the scope of possibilities for generating the openings. The aim was to then perform an analysis based on the desired design metrics for every possibility within this scope, identify the best ones and allow the designer to choose between these with the knowledge as to their performance, but this was not implemented and instead random variations within the scope were generated and chosen.

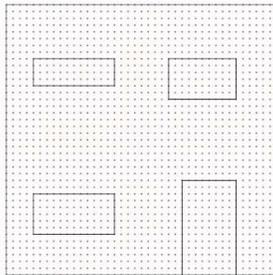


Figure 4.1-2 : Step 1

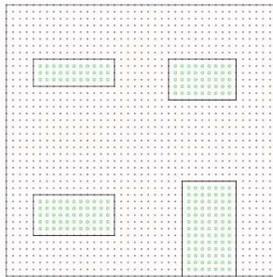


Figure 4.1-3 : Step 2

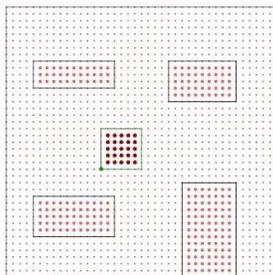


Figure 4.1-4 : Step 3

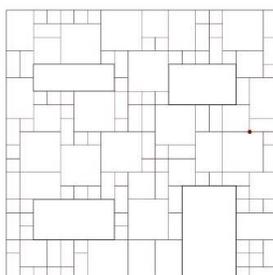


Figure 4.1-5 : Final result

The next step concerned the cladding of the facades. To achieve the previously described desired result, an algorithmic logic was developed to generate an efficient tiling with various sized tiles (based on multiples of the unit tile). The algorithm functioned as follows:

1. Generate the grid of points based on the unit tile size (Figure 4.1-2)
2. Determine which points lie within the generated openings and form a list of exclusion points (Figure 4.1-3)
3. Choose a random point:
 - From this point form a rectangle of the current size in point multiples (starting at a determined max size)
 - Verify that no points within it are part of the exclusion point list (Figure 4.1-4)
 - If ok → go to step 4
 - If not → choose a new random point
 - If Nth attempt, go down a rectangle size
4. Add the rectangle to the output rectangle list
5. Add the internal and perimeter points to the exclusion point list
6. Go back to step 3.

This is of course a simplification of the final logic but provides an accessible and honest overview as to its functioning. Given the cyclical nature of this algorithm, in comparison to the linear nature of parametric design and visual programming, it was implemented using scripting, originally thanks to the use of the python script component and the RhinoScript²³ framework.

Further logic then followed which allowed the attribution of a random color within a predefined palette to each tile, which could also be explored by the designer.

²³ <https://developer.rhino3d.com/guides/rhinoscript/>

4.1.3.Implementations

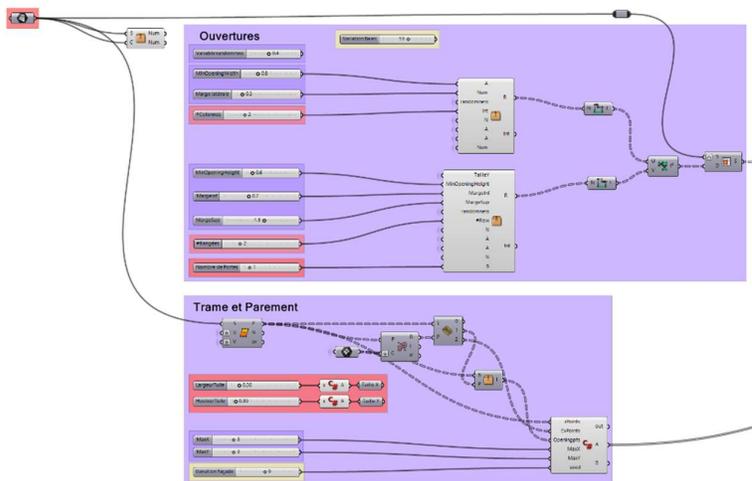
As the original script contained other aspects and had not been created with reuse in mind, the steps for generating openings, generating the cladding, and exploring variations were extracted and adapted in the following ways:

- Where the original script predetermined the number of openings, **parameters to choose the number of rows, number of columns, and number of doors in the openings were implemented.**
- Although it would have been possible to implement the analysis step, given the vague design context and **to keep script execution as quick and fluid as possible**, the choice was made to function through the **exploration of random variations**, but still **within a design scope** that can be manipulated.
- **The cladding algorithm was completely rewritten and re implemented.** Originally created in a python script component using RhinoScript, it was rewritten and optimized (in part) in C# using RhinoCommon. This allowed **easier manipulation of the logic**, as well as **substantial performance benefits**, going from an average execution time of over 1 minute to under 50ms. It also facilitated the implementation of the Rhino command.
- The **generation of window and door geometry** was also **refactored and optimized** for performance.

The resulting logic was then reimplemented to form variants for each of the contexts that were the basis of the study. These implementations as well as the template file can be found on the GitHub repository²⁴ of this study.

The details of each implementation are further explained in the following paragraphs.

- GH context: For this variant, the regular best practice rules for grasshopper script structuring were followed: grouping the various steps of the logic, clearly identifying the parameters, and labeling where necessary (Figure 4.1-6).



The subjects were instructed to interact freely with the script; other than explicit labeling and grouping, as well as using clusters for more complex sub-logic, no additional efforts were made to render the script more accessible. The parameters could be manipulated in the context of the parts of the script where they are implemented. As such, the user was exposed to the entire script.

Figure 4.1-6 : Grasshopper implementation view

²⁴ <https://github.com/XGar/Impact-of-context-familiarity-on-computational-design-logic-appropriation>

- Hybrid context:

The aim of this variant is to remain in the typical rhino workspace for the interaction but use methods of interaction that are not typical to ordinary Rhino workflows, and still have access to the grasshopper script if finer adjustment or control is needed.

The geometry is automatically referenced based on layer placement.

Parameters such as grid spacing and limits can be manipulated thanks to contextual sliders in the rhino viewport that are linked to the geometry (à droite dans la fig xxx). This is achieved using the kangaroo physics engine.

The other parameters are manipulated through overlaid controls, thanks to the use of HumanUI (seen on the left on Figure 4.1-7).

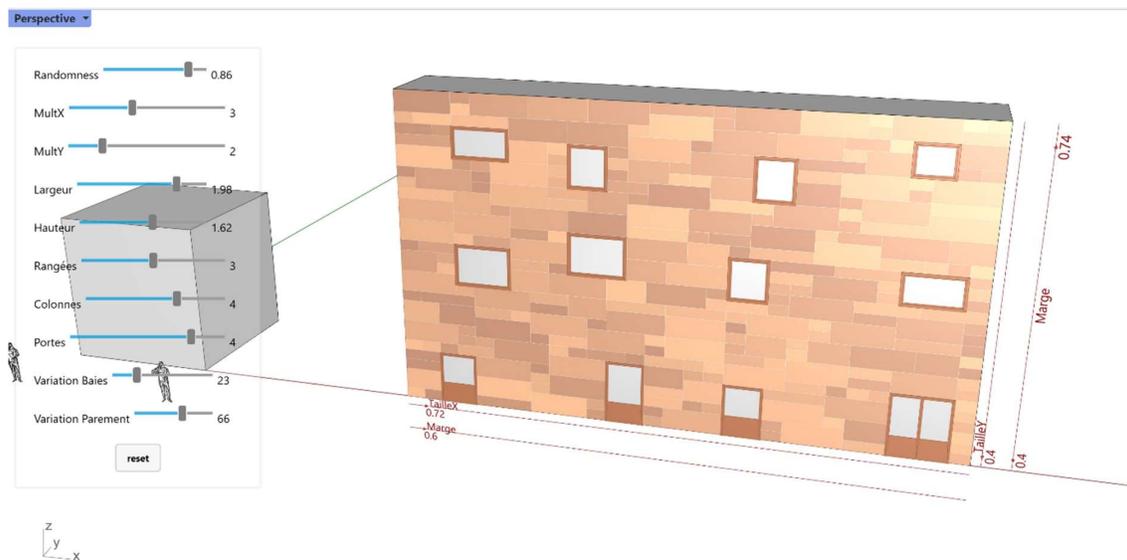


Figure 4.1-7 : Hybrid implementation view

- Rhino Context:

This implementation aims to replicate a typical rhino command in terms of interaction. The entire logic was therefore recreated through C# programming and compiled into a plugin. The interaction is done entirely through the command line and the rhino viewport as with any other command.

After calling the command, the subject is prompted to select the target surface. The principal menu then appears in the command line (Figure 4.1-9) and the subject can navigate through 2 submenus depending on whether they would like to interact with parameters that determine the grid settings and cladding variations (Figure 4.1-10) or parameters that define and explore the openings of the façade (Figure 4.1-11).

```
Command: Facade
Please select the target surface

( Maillage Baies ):
```

Figure 4.1-9 : Principal Menu

```
Command: Facade
Please select the target surface
( Maillage Baies ): Maillage

Taille ( X=0.3 Y=0.3 Random_Next Random_Back Multiple_X=3 Multiple_Y=3 ):
```

Figure 4.1-10 : Grid submenu

```
Command: Facade
Please select the target surface
( Maillage Baies ): Maillage
Taille ( X=0.3 Y=0.3 Random_Next Random_Back Multiple_X=3 Multiple_Y=3 )
( Maillage Baies ): Baies

Baies: ( Rangées=2 Colonnes=2 randomness=0.2 Random_Next Random_Back Marge_Laterale=0.2 HauteurAllège=0.8 MargeSup=0.2 LargeurMin=0.8 HauteurMin=0.8 NbrPorte=1 ):
```

Figure 4.1-11 : Opening submenu

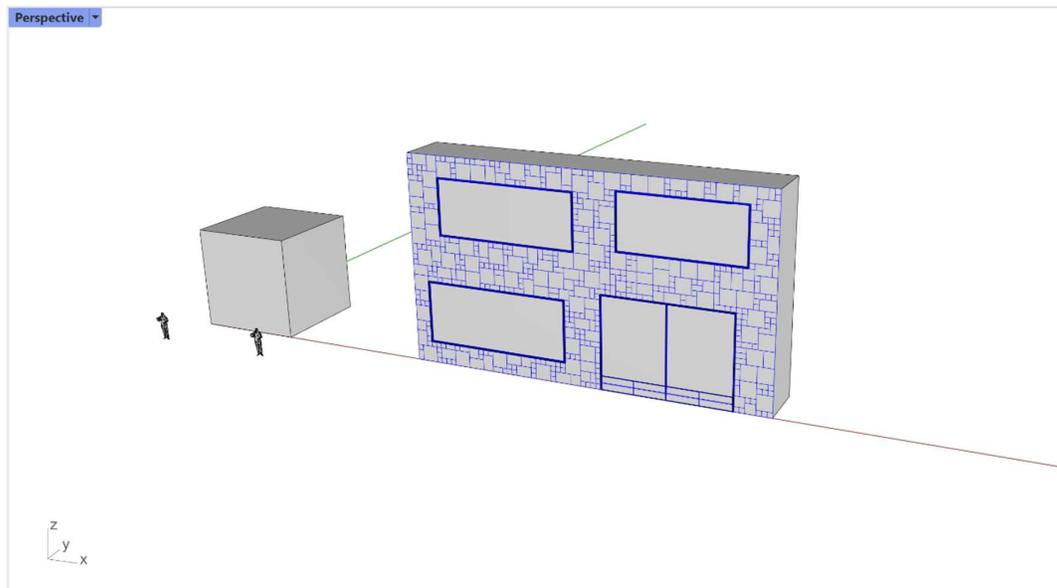


Figure 4.1-8 : Result preview

4.2. User selection

For this study, the test subjects were all from the architectural engineering section of the Faculty of Applied Science of the University of Liège and were either students or recent graduates (less than one year of professional experience). This decision was made for three reasons:

- To ensure the skills and experience consistency of the tested population in terms of digital practices and software knowledge.
- To ensure their relative homogenous background in terms of approach to architectural design.
- To present a variety of proficiency levels, but with a common foundation provided by the introductory course in Rhino and grasshopper given in the third year.

While a more diverse subject pool could also have produced interesting results, since part of the objective was to test the common assertion that lack of familiarity in a given software is a major obstacle to the use of computational design, the choice was made to conduct the study within the bounds of this described subject pool. While some subjects were still students and others already working, the differences between the two were not explicitly studied, the hypothesis being that given the maximum of one year of professional experience by some of the subjects, this difference would have less impact than their differing software proficiencies, although these are undoubtedly linked. A possible later development of this research which would be interesting variation could be to conduct this experiment among subjects of various ages or at various stages of their professional career, to see how cultural differences between generations and varying mindsets impact the interactions.

4.3. Experiment Protocol

In this section, we will describe the how we structured the experiment protocol in order to ensure the harvesting of data in a **coherent, reliable, and rigorous manner** (Calixte X. , 2021).

Given the difficulties in performing a systematic qualitative and categorical analysis of the use of tools for design processes without a more robust protocol (Calixte, Rajeb, & Leclercq, 2018), we instead chose to perform a quantitative analysis.

We conducted multiple surveys with two types of questions; grid/numerical based questions to ensure a consistency in the form of the answers to base a quantitative analysis upon, and open questions which allowed us to contextualize and inform the conclusions of this analysis.

We also chose to study the interactions with the tools by limiting the focus to the actions on these tools (Calixte X. , 2021). To do this systematically, we implemented a data harvesting strategy to take into account the entirety of the actions that concerned these tools (Otjacques, 2008).

4.3.1. Experiment environment variables

The experiment was conducted between the December 20, 2021, and December 28, 2021, either at the University or in my home, to accommodate the schedules and preferences of both the students and the recent graduates.

For each experiment, the entire interaction took place using my computer to remove obstacles due to licensing issues and plugin compatibility, as well as to implement automatic data harvesting more easily.

The subject was seated at a table in front of the computer, with a mouse and keyboard to interact with it, and myself seated next to them to provide instructions as well as help if needed.

4.3.2. Subject Characterization

Although subjects having passed the third-year course in the ULiege engineer-architect curriculum were introduced to Rhino, Grasshopper, and computational design²⁵, their familiarity with these topics varied as their subsequent use of these programs is not systematic and the course introducing them to these subjects has also evolved. There is also a variability in general concerning modeling program preferences and proficiency. To gauge their current level and overall familiarity with Rhino, Grasshopper, and computational design thinking, as well as gain insight as to their current modeling habits and preferred tools, the first step of the experiment was to conduct a survey to collect information for establishing profiles. Both open and closed questions are asked, as described below:

- *What is your experience with computational design?*
(None | Theoretical | Use of existing logic | Creation of custom logic)
- *In which circumstances have you previously used or created computation design logic?*
(Introductory course | Project | Other)
- *According to you, what is the use of computational design and where is it applicable?*
(Open question)
- *What software have you used and at what proficiency level would you say you are?*
(Software proficiency survey with AutoCAD, Blender, SketchUp, Rhino, Grasshopper, Revit, Archicad and 3dsMax as software choices and No experience, Novice, Limited, Basic, Advanced and Expert as possible proficiency levels) (Figure 4.3-1)

5. Quels logiciels utilisez vous et ou situez-vous votre niveau?

	Aucune expérience	Novice	Limité	Habitué	Avancé	Expert
Autocad	<input type="radio"/>					
Blender	<input type="radio"/>					
SketchUp	<input type="radio"/>					
Rhino	<input type="radio"/>					
Grasshopper	<input type="radio"/>					
Revit	<input type="radio"/>					
Archicad	<input type="radio"/>					
3dsMax	<input type="radio"/>					

Figure 4.3-1 : Software proficiency survey

- *What is your experience with and in what context have you used Rhino and Grasshopper specifically?* (Open question)
- *What currently prevents you from using computational design more often?* (Open question)

²⁵ Details on course can be found here:

<https://www.programmes.uliege.be/cocoon/20212022/cours/ARCH0017-4.html>

4.3.3. Tool interactions

During this step of the experiment, the subject interacted with the design logic reimplemented in the three different contexts as described in section 4.1.3. The order in which they interacted with the different implementations was determined based on the order in which the different subjects did the study. In this way a uniform distribution of the different combinations of order of interaction possible could be approached to limit the impact of the order of interaction, as studied in section 5.3. The origin and description of the general design logic were presented to the subject, followed by details of each implementation that they were about to interact with. For each implementation the starting setup as well as the provided design scenario is the same. Based on this design scenario, the subject was instructed to interact with the implementation until they were satisfied with the result. Each interaction or variation was automatically timestamped and logged for later analysis (the data harvesting strategy is described in section 4.4).

4.3.4. Subject feedback

After each of the three exercises, the subject was asked to continue the survey in order to collect their feedback on the interaction method, as well as their satisfaction with the final result.

They were asked to answer the questions using a scale from 0 (not at all) to 10 (very much):

- Accessibility: How easy was it to start interacting with this implementation, having no previous experience with it?
- Comprehension: How easy was it to understand the underlying design logic by using this implementation?
- Ease of Use: How easy was it to interact with the design logic using this implementation?
- Usability: To what extent could you see yourself interacting with computational design logic in this way in your day-to-day work?
- Satisfaction: How satisfied were you in your ability to achieve your desired outcome through this implementation?

An open question was also asked to allow them to freely express their feedback on their overall experience. They also had the possibility to ask additional questions if needed. At the end of the experiment, the subjects were again asked to rate *a posteriori* the implementations in the different contexts, with the experience of having done them all. The complete experiment sequence can be seen on Figure 4.3-2.

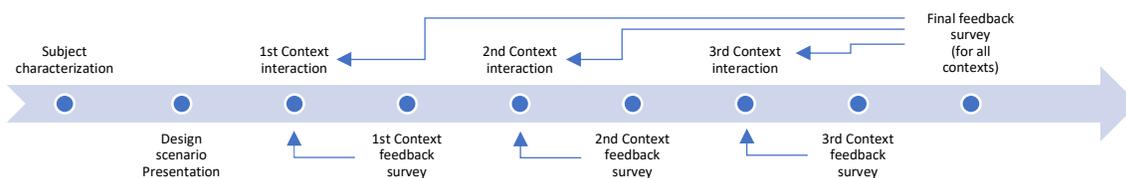


Figure 4.3-2 : Experiment sequence

4.3.5. Protocol validation

To validate the protocol, a test experiment was conducted before starting any interviews. This “experiment 0” showed that the initial time estimate for the entire duration of the experiment (20 minutes) was too short and had to be adjusted to 1 hour per subject on average. This is one of the reasons why the subject sample size was not bigger. This test experiment also revealed several bugs in the Rhino and Hybrid implementations.

4.4. Data harvesting

To shed light on the research question, two main sources of harvested data were analyzed during the experiment.

The first source of data were the surveys conducted during the experiment in order to profile and characterize the subjects (their proficiency level in different software mostly) and to collect their feedback on the interactions (both immediately following each context as well as after having done all three). Given that this survey was conducted through Microsoft Forms, a *.csv file containing all the data could be easily exported and manipulated for the analyses.

The second relevant datapoint pertained to how the subject interacted with each tool implementation. To analyze the subject's behavior during the interaction, a bespoke tool was developed: for each modification of any parameter, a line was automatically added to a csv file. It contained a timestamp, an identifier as to the current implementation and geometry (which façade) concerned, and the values of each of the 15 different parameters in a constant order (Figure 4.4-1). This allowed the automatic harvesting of substantial amounts of data that would not have been feasible otherwise.

Time	Object	TileXSize	TileYSize	Variability	Columns	Rows	MultipleX	MultipleY	CladdingSeed	OpeningSeed	MargeX	UpperMargin	LowerMargin	MinOpeningHeight	MinOpeningWidth	DoorNumber
12:17:59.254	3.1	0.3	0.3	0.2	2	2	3	3	1	1	0.2	0.2	0.8	0.8	0.8	1
12:18:17.405	3.1	0.4	0.3	0.2	2	2	3	3	1	1	0.2	0.2	0.8	0.8	0.8	1
12:18:29.512	3.1	0.38	0.3	0.2	2	2	3	3	5	1	0.2	0.2	0.8	0.8	0.8	1
12:18:30.228	3.1	0.38	0.3	0.2	2	2	3	3	9	1	0.2	0.2	0.8	0.8	0.8	1

Figure 4.4-1 : Data harvesting example

This tool is published on GitHub²⁶ and easily accessible and usable to any researcher or future TFE student who would like to conduct a similar experiment with Grasshopper or Rhino.

The fact that all the data was harvested automatically allowed all the experiments to be led in a brief time span without the additional strain of having to harvest the data manually at the same time, and reduced the chance of human error. This also allowed more time to be invested on data cleaning, engineering, and visualization, and thus focus on developing a more robust analysis framework that could be built upon.

²⁶ <https://github.com/XGar/Impact-of-context-familiarity-on-computational-design-logic-appropriation>

4.5. Data Cleaning

In this section, we will present the steps and strategies that were employed to operate the data cleaning on the raw data. This data cleaning was all done through the use of a Jupyter²⁷ Notebook and Python²⁸, using the Numpy²⁹ and Pandas³⁰ libraries, in DataSpell³¹. The methods of data cleaning, data engineering and data visualization were learned during the course of this research and through various online resources, mainly the Numpy and Pandas API references, as well as by following several free courses found on freeCodeCamp^{32,33} and a paid course³⁴ on the Udemy platform. All the raw³⁵ data and source code are available on the GitHub repository³⁶ of the project. Rather than performing a detailed code walkthrough, we will provide a high-level overview of the structure of the implemented code to explain the way in which the data was assembled and transformed.

4.5.1. Survey data cleaning

Once harvested, the data had to be cleaned and processed. For the survey data, the data pertaining to the subjects' answers for their software proficiency was extracted from the csv that was exported from Microsoft Forms and formed into a Pandas DataFrame³⁷ with a column for each software and a row for each subject. As only two of the participants had (limited) experience in Blender and 3dsMax, these categories were removed from the DataFrame. Additional columns were then added:

- An average column: the average software proficiency for each subject, disregarding the software in which they had no experience. This was initially done to not overly penalize subjects who only had experience in a couple of different software, although this did penalize those who had slight experience in more software and as a whole this was not a representative metric and unused for the analysis in section 5.4.
- A column representing the subject's level in the "traditional" CAD software most used in the AEC field in Belgium currently (Stals, Elsen, & Jancart, Practical Trajectories of Parametric Tools in Small and Medium Architectural Firms, 2017): AutoCAD and SketchUp. This indicator was obtained by keeping the maximum value between the AutoCAD and SketchUp results for all the subjects.
- A column representing the subject's experience in BIM modeling, obtained by keeping the maximum value between the Revit and Archicad results.
- A column representing the subject's average experience in Rhino and Grasshopper, as an indicator to analyze the Hybrid context implementation within the subject's existing experience.

In an analogous way, the data regarding the subject's feedbacks on the interactions were grouped into two lists of three DataFrames: one for the answers immediately following each interaction, and one for the answers given during the final survey. An example can be seen on Figure 4.5-1.

²⁷ <https://jupyter.org/>

²⁸ At the time of development, Python 3.9.7 <https://www.python.org/>

²⁹ <https://numpy.org/doc/stable/reference/index.html#reference>

³⁰ <https://pandas.pydata.org/docs/reference/index.html>

³¹ JetBrains DataSpell IDE : <https://www.jetbrains.com/dataspell/>

³² <https://www.freecodecamp.org/news/python-data-science-course-matplotlib-pandas-numpy/>

³³ <https://www.freecodecamp.org/news/data-analysis-with-python-for-excel-users-course/>

³⁴ <https://www.udemy.com/course/python-for-machine-learning-data-science-masterclass/>

³⁵ The original data contained the subjects' names, those names were replaced by identifiers by passing the raw harvested data through a data obfuscation function.

³⁶ <https://github.com/XGar/Impact-of-context-familiarity-on-computational-design-logic-appropriation>

³⁷ A Pandas DataFrame is a data structure based on Numpy and widely used for data science.

Nom	Prise en main	Compréhension/maitrise de la logique sous-jacente	Fluidité d'interaction	Potentiel d'utilisation au quotidien/d'intégration dans des workflows existants	Satisfaction du résultat final
Subject1	7	8	7	10	8
Subject2	10	10	8	7	8
Subject3	10	8	5	6	7
Subject4	8	8	6	8	6
Subject5	5	6	8	8	8
Subject6	9	7	7	2	7
Subject7	10	10	10	10	10
Subject8	9	9	9	8	8
Subject9	10	8	5	10	3
Subject10	7	8	7	8	7
Subject11	10	8	10	9	9
Subject12	9	8	7	10	9
Subject13	6	10	7	10	9
Subject14	6	8	7	4	7

Figure 4.5-1 : Feedback DataFrame example (1st interaction survey)

4.5.2. Tool interactions data cleaning

For each of the subjects, a DataFrame was constructed from the raw data and the timestamps correctly formatted. The data was then split by implementation, the data for the façade used to explain the implementation was pruned, and the order in which the subject had done the experiments was identified and used to reorder the experiment data as well as the feedback data in order to have the data in a consistent order based on the three contexts (Grasshopper, Hybrid, Rhino).

4.6. Data Engineering

In this section, we will provide an overview of how the cleaned and assembled data from the previous step was transformed to create the various DataFrames that will be used for the analysis. The tools and methods used were the same as explained in section 4.5.

For this research, the tool interactions were analyzed under the following three indicators which were created from the harvested data:

- **Number of iterations:** From this indicator, we will focus on how many times and when each of the parameters were changed and as such another iteration of the script was computed. The detail of how this was deduced and put into form from the data will be the subject of section 0.
- **Modification phases:** The point of interest for this indicator is the study of when the subjects passed from changing one parameter to another. We will track these changes in order to know for each parameter the number of phases in which it was manipulated, as well as to know how this behavior evolved over the course of the interaction. The creation of the data needed for this analysis will be explained in section 0.
- **Unique values:** This analysis indicator focuses on the number of unique values explored by the subject for each parameter and where in the timeline these numbers evolve. The specificities involved for determining this from the input data and how it was implemented will be presented in section 4.6.3.

Given the number of DataFrames that will be used and referenced, and the number of transformations applied to the input data over several step, Table 1 was created to provide an overview of these DataFrames.

Table 1: DataFrame overview

Name	Description	input	Figure	Sections used
imp_df	Raw data for the façade	Csv	Figure 4.6-3	4.6.1, 4.6.3
diff_df	Boolean DataFrame representing if a value of imp_df has changed from the previous row.	imp_df	Figure 4.6-2, Figure 4.6-4	4.6.1, 4.6.2
diff_diff_df	Boolean DataFrame representing if a value from diff_df has changed from the previous row.	diff_df	Figure 4.6-5	4.6.2
iterations_df	DataFrame representing the number of times a parameter has been changed	diff_df	Figure 4.6-1	4.6.1
time_iterations_df	Evolution of diff_df over time	diff_df	xxx	4.6.1
phase_df	Number of distinct sequences during which the subject manipulated a given parameter	diff_diff_df	Figure 4.6-6	4.6.2
time_phase_df	Evolution of diff_diff_df over time	diff_diff_df	xxx	4.6.2
unique_df	DataFrame representing the number of unique values of a parameter in imp_df	imp_df	Figure 4.6-8	4.6.3
time_unique_df	DataFrame representing the evolution of the number of unique values over time	imp_df	xxx	4.6.3

4.6.1. Iterations data engineering

The main iterations DataFrame (iterations_df) simply represents how many times a given parameter was changed. Data was organized by façade and tool implementation (imp_df). A boolean dataframe diff_df indicating whether a value in imp_df is different to the one in the row above is created to identify which parameters were changed. The sum of the columns diff_df formed a row iterations_df upon which the implementation, façade and time were added as additional columns or indexes.

Time	Object	TileXSize	TileYSize	Variability	Columns	Rows	MultipleX	MultipleY	AddingSee	OpeningSee	MargeX	pperMarge	owerMarge	OpeningHe	OpeningW	WoorNumbe
00:00:00	2	0,3	0,3	0,2	2	2	3	3	1	1	0,2	0,2	0,8	0,8	0,8	1
00:00:07	2	0,3	0,3	0,2	2	3	3	3	1	1	0,2	0,2	0,8	0,8	0,8	1
00:00:10	2	0,3	0,3	0,2	3	3	3	3	1	1	0,2	0,2	0,8	0,8	0,8	1
00:00:15	2	0,3	0,3	0,3	3	3	3	3	1	1	0,2	0,2	0,8	0,8	0,8	1
00:00:32	2	0,3	0,3	0,3	3	3	3	3	1	1	0,4	0,2	0,8	0,8	0,8	1
00:00:37	2	0,3	0,3	0,3	3	3	3	3	1	1	0,4	0,2	1,1	0,8	0,8	1
00:00:41	2	0,3	0,3	0,3	3	3	3	3	1	1	0,4	0,2	1,1	0,8	0,8	1
00:00:45	2	0,3	0,3	0,3	3	3	3	3	1	2	0,4	0,2	1,1	0,8	0,8	2
00:00:46	2	0,3	0,3	0,3	3	3	3	3	1	3	0,4	0,2	1,1	0,8	0,8	2
00:00:47	2	0,3	0,3	0,3	3	3	3	3	1	4	0,4	0,2	1,1	0,8	0,8	2
00:00:47	2	0,3	0,3	0,3	3	3	3	3	1	5	0,4	0,2	1,1	0,8	0,8	2
00:00:48	2	0,3	0,3	0,3	3	3	3	3	1	6	0,4	0,2	1,1	0,8	0,8	2
00:00:49	2	0,3	0,3	0,3	3	3	3	3	1	5	0,4	0,2	1,1	0,8	0,8	2
00:01:01	2	0,5	0,3	0,3	3	3	3	3	1	5	0,4	0,2	1,1	0,8	0,8	2
00:01:06	2	0,5	0,7	0,3	3	3	3	3	1	5	0,4	0,2	1,1	0,8	0,8	2
00:01:16	2	0,5	0,67	0,3	3	3	3	3	1	5	0,4	0,2	1,1	0,8	0,8	2
00:01:33	2	0,5	0,67	0,3	3	3	3	3	1	5	0,4	0,2	2	0,8	0,8	2
00:01:53	2	0,5	0,67	0,3	3	3	3	3	1	5	0,4	0,2	2	0,8	0,8	2
00:02:13	2	0,5	0,67	0,3	3	3	3	3	1	5	0,4	0,2	1,1	0,8	0,8	1
00:02:17	2	0,5	0,67	0,3	3	3	3	3	1	5	0,4	0,2	0,8	0,8	0,8	1

Figure 4.6-3 : imp_df example

Time	Object	TileXSize	TileYSize	Variability	Columns	Rows	MultipleX	MultipleY	AddingSee	OpeningSee	MargeX	pperMarge	owerMarge	OpeningHe	OpeningW	WoorNumbe
00:00:00	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:00:07	2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
00:00:10	2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
00:00:15	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
00:00:32	2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
00:00:37	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
00:00:41	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:00:45	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
00:00:46	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
00:00:47	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
00:00:47	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
00:00:48	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
00:00:49	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
00:01:01	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:01:06	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
00:01:16	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
00:01:33	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
00:01:53	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:02:13	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
00:02:17	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Figure 4.6-2 : diff_df example

	Object	TileXSize	TileYSize	Variability	Columns	Rows	MultipleX	MultipleY	AddingSee	OpeningSee	MargeX	pperMarge	owerMarge	OpeningHe	OpeningW	WoorNumbe
0	40	1	2	1	1	1	0	0	0	6	1	0	4	0	0	2

Figure 4.6-1 : iterations_df example

A second DataFrame represents the evolution of this data over time. Rather than calculate the sum for the entire façade, in this second DataFrame the data for each façade is resampled at regular intervals of the total time spent on the given façade. Which means that the datapoints that fell within that time range were added along. The chosen number of divisions to define the interval is ten, and while a higher number of divisions would have given a higher resolution, ten divisions appeared to be enough, as some intervals (usually the last one) occasionally had no data points.

4.6.2. Modification phases

A modification phases DataFrame (phase_df) was constructed by using a similar logic. For each action recorded, the parameter values are compared to the previous ones, allowing to identify if there was a change, e.g. if the subject manipulated it (either through sliders or command lines according to the tool implementation)

Time	Object	TileXSize	TileYSize	Variability	Columns	Rows	MultipleX	MultipleY	AddingSee	PenningSee	MargeX	pperMarge	owerMarge	OpeningHe	OpeningWoor	Number	
00:00:00	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:00:07	2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
00:00:10	2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
00:00:15	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
00:00:32	2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
00:00:37	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
00:00:41	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:00:45	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
00:00:46	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
00:00:47	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
00:00:47	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
00:00:48	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
00:00:49	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
00:01:01	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:01:06	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:01:16	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:01:33	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
00:01:53	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:02:13	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
00:02:17	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 4.6-4 : diff_df example

Time	Object	TileXSize	TileYSize	Variability	Columns	Rows	MultipleX	MultipleY	AddingSee	PenningSee	MargeX	pperMarge	owerMarge	OpeningHe	OpeningWoor	Number	
00:00:00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:00:07	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
00:00:10	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
00:00:15	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
00:00:32	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
00:00:37	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
00:00:41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:00:45	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
00:00:46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:00:47	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:00:47	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:00:48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:00:49	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:01:01	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:01:06	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:01:16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:01:33	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
00:01:53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00:02:13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
00:02:17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4.6-5 : diff_diff_df example

	Object	TileXSize	TileYSize	Variability	Columns	Rows	MultipleX	MultipleY	AddingSee	PenningSee	MargeX	pperMarge	owerMarge	OpeningHe	OpeningWoor	Number
0	0	1	1	1	1	1	0	0	0	1	1	0	3	0	0	2

Figure 4.6-6 : phase_df example

The resulting DataFrames for each façade of each implementation was then either summed along the columns to form a row of the number of modification phases for each façade-implementation combination and added to the modification phases DataFrame, or in the same way as for the iterations a DataFrame of the evolution of the modification phases over time (time_phase_df) was constructed by resampling the DataFrames from the previous step across regular time range intervals.

4.6.3. Unique values

The unique values DataFrame was constructed using the `nunique`³⁸ function in pandas which returns the number of unique values of each column. This means that when studying the number of unique values, we are currently only looking parameter by parameter and not the combinations. This shows a very high number of unique parameters explored in each experiment. While this informs us that the data collection was effective, it was not significant of the studied interactions. The choice was therefore made to not implement³⁹ it and to analyze the results parameter by parameter.

Time	Object	TileXSize	TileYSize	Variability	Columns	Rows	MultipleX	MultipleY	AddingSe	OpeningSec	MargX	pperMarg	owerMarg	OpeningHe	OpeningW	oorNumbe
00:00:00	2	0,3	0,3	0,2	2	2	3	3	1	1	0,2	0,2	0,8	0,8	0,8	1
00:00:07	2	0,3	0,3	0,2	2	3	3	3	1	1	0,2	0,2	0,8	0,8	0,8	1
00:00:10	2	0,3	0,3	0,2	3	3	3	3	1	1	0,2	0,2	0,8	0,8	0,8	1
00:00:15	2	0,3	0,3	0,3	3	3	3	3	1	1	0,2	0,2	0,8	0,8	0,8	1
00:00:32	2	0,3	0,3	0,3	3	3	3	3	1	1	0,4	0,2	0,8	0,8	0,8	1
00:00:37	2	0,3	0,3	0,3	3	3	3	3	1	1	0,4	0,2	1,1	0,8	0,8	1
00:00:41	2	0,3	0,3	0,3	3	3	3	3	1	1	0,4	0,2	1,1	0,8	0,8	1
00:00:45	2	0,3	0,3	0,3	3	3	3	3	1	2	0,4	0,2	1,1	0,8	0,8	2
00:00:46	2	0,3	0,3	0,3	3	3	3	3	1	3	0,4	0,2	1,1	0,8	0,8	2
00:00:47	2	0,3	0,3	0,3	3	3	3	3	1	4	0,4	0,2	1,1	0,8	0,8	2
00:00:47	2	0,3	0,3	0,3	3	3	3	3	1	5	0,4	0,2	1,1	0,8	0,8	2
00:00:48	2	0,3	0,3	0,3	3	3	3	3	1	6	0,4	0,2	1,1	0,8	0,8	2
00:00:49	2	0,3	0,3	0,3	3	3	3	3	1	5	0,4	0,2	1,1	0,8	0,8	2
00:01:01	2	0,5	0,3	0,3	3	3	3	3	1	5	0,4	0,2	1,1	0,8	0,8	2
00:01:06	2	0,5	0,7	0,3	3	3	3	3	1	5	0,4	0,2	1,1	0,8	0,8	2
00:01:16	2	0,5	0,67	0,3	3	3	3	3	1	5	0,4	0,2	1,1	0,8	0,8	2
00:01:33	2	0,5	0,67	0,3	3	3	3	3	1	5	0,4	0,2	2	0,8	0,8	2
00:01:53	2	0,5	0,67	0,3	3	3	3	3	1	5	0,4	0,2	2	0,8	0,8	2
00:02:13	2	0,5	0,67	0,3	3	3	3	3	1	5	0,4	0,2	1,1	0,8	0,8	1
00:02:17	2	0,5	0,67	0,3	3	3	3	3	1	5	0,4	0,2	0,8	0,8	0,8	1

Figure 4.6-7 : imp_df

	Object	TileXSize	TileYSize	Variability	Columns	Rows	MultipleX	MultipleY	AddingSe	OpeningSec	MargX	pperMarg	owerMarg	OpeningHe	OpeningW	oorNumbe
0	1	2	3	2	2	2	1	1	1	6	2	1	3	1	1	2

Figure 4.6-8 : unique_df

To construct the DataFrame representing the evolution of unique values over time, almost the same logic as for the iterations and modification phases was applied. The main difference was that instead of calculating the number of unique values for the data of each time interval, the number of new unique values had to be calculated by subtracting the number of unique values from the beginning of the façade to the beginning of the interval from the number of unique values from the beginning of the façade to the end of the interval.

For each row of each of the main DataFrames, a multi-index was added containing the corresponding subject's identifying number, subject's proficiency in the different software (concatenated in a number), the order in which the contexts were done, the context, and the façade each row refers to. For each subject, rows were added for the total values across the façades, as well as their average and standard deviation (Figure 4.6-9). They were then exported in a csv format.

Name	Level	Order	Context	Object	TileXSize	TileYSize	Variability	Columns	Rows	...	DoorNumber	Time	Max	Total	CV
Subject1	3311262321	1	Grasshopper	2	4.00	4.00	0.00	14.00	8.00	...	5.0	188.00	34.00	121.00	1.07
				3	2.00	0.00	0.00	5.00	6.00	...	3.0	144.00	18.00	91.00	1.10
				4	0.00	0.00	0.00	2.00	9.00	...	2.0	75.00	9.00	20.00	2.11
				2	0.00	0.00	3.00	0.00	1.00	...	3.0	141.00	5.00	24.00	1.00
		2	Hybrid	3	2.00	0.00	1.00	0.00	0.00	...	2.0	53.00	2.00	10.00	1.22
				4	2.00	0.00	0.00	1.00	0.00	...	1.0	88.00	5.00	11.00	1.82
				2	0.00	0.00	0.00	0.00	0.00	...	1.0	73.00	5.00	9.00	2.25
				3	0.00	0.00	1.00	0.00	0.00	...	1.0	85.00	10.00	24.00	1.92
		3	Rhino	4	0.00	0.00	0.00	1.00	2.00	...	0.0	21.00	2.00	3.00	2.00
				2	4.00	4.00	3.00	14.00	9.00	...	9.0	402.00	42.00	154.00	0.97
				3	4.00	0.00	10.00	5.00	6.00	...	6.0	282.00	23.00	125.00	0.89
				4	2.00	0.00	0.00	4.00	11.00	...	3.0	182.00	11.00	34.00	1.51
		0	Tot	2	1.33	1.33	1.00	4.67	3.00	...	3.0	134.00	14.00	51.33	0.97
				3	1.33	0.00	3.33	1.67	2.00	...	2.0	94.00	7.67	41.67	0.89
				4	0.67	0.00	0.00	1.33	3.67	...	1.0	60.67	3.67	11.33	1.51
				2	2.31	2.31	1.73	8.08	4.36	...	2.0	57.82	17.35	69.66	0.89
		Standard Deviation	3	1.15	0.00	4.04	2.89	3.46	...	1.0	46.16	10.12	55.03	0.98	
			4	1.15	0.00	0.00	0.58	4.73	...	1.0	35.16	4.73	14.71	1.58	

Figure 4.6-9 : Main DataFrame example

³⁸ <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.nunique.html>

³⁹ This could be done by simply concatenating each row into a string with a separator thanks to the `str.cat` function, then using the `nunique` function to obtain the number of unique combinations of parameters.

4.7. Data visualization

Once the data was cleaned and transformed, the next step was to provide a way to easily analyze the results. Given the large amount of data as well as all the diverse ways we wanted to combine and cross reference the data, data visualization was used instead of comparing numbers. This was helpful to easily and quickly detect which datapoints were significant and would require more in-depth analysis or discussion.

Care was taken to work in a way that would easily accept more data and could be easily adapted to study certain aspects more in depth or according to specific parameters or statistical tools. Therefore, a framework was developed that allowed to automatically generate “Dashboards” based on the previous indicators defined in section 4.6. These dashboards combine several types of graphs to provide an overview of the impact of a chosen factor (such as proficiency in a certain software, or the order in which the experiment was done) on either the average interactions or those in a specific context.

In practice, this was done using a Jupyter Notebook in python using JetBrains’s DataSpell IDE, with extensive use of the Pandas, Numpy, Matplotlib and Seaborn libraries. The use of a Jupyter Notebook allowed easier development of the functions that enabled the generation of these dashboards. Once these functions were deemed sufficiently developed, they were grouped in a separate python file that could be referenced as a custom package, after which the Jupyter Notebook was used mainly for exploring and generating the different dashboards, as well as exporting them to be saved. As for the previous development done for this research, this code is available on GitHub⁴⁰ to future researchers.

For the analysis, two main functions were created and used to generate the dashboards: ***feedback*** (described in section 0) and ***global_analysis*** (section 4.7.2).

⁴⁰ <https://github.com/XGar/Impact-of-context-familiarity-on-computational-design-logic-appropriation>

4.7.1.feedback

The feedback function defined in the analysis package⁴¹ can be called in the following way with these arguments:

feedback(feedback_df, context_index, software, survey, _level)

- **feedback_df**: the pandas DataFrame of the subjects' feedbacks concerning the interactions generated by the output csv from the data engineering step.
- **context_index**: this allows the study of either a specific context (GH=0, Hybrid=1, Rhino=2), the average result (4) or if None is passed the function will plot all the results (so three for each subject).
- **software**: this is the software that will be used to group the subjects by proficiency level if that is the chosen study variable.
- **survey**: this allows the study of the survey conducted immediately following a given context (A), at the end of the experiment (B), or both side by side (None).
- **_level**: this is the study variable with which the surveys are examined, either by proficiency level ("level"), by order ("order") or by context ("context").

The output of this function is a bar plot⁴² as in *Figure 4.7-1*, where for each category of the feedback survey the results are grouped by the study variable, in this case by grasshopper level, with the individual data points overlaid via a swarm plot⁴³.

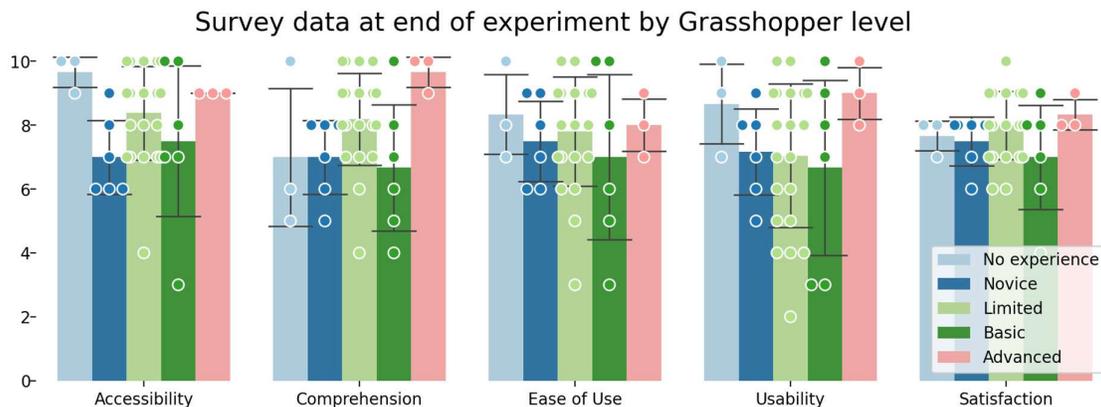


Figure 4.7-1 : feedback graph example

⁴¹ <https://github.com/XGar/Impact-of-context-familiarity-on-computational-design-logic-appropriation>

⁴² <https://seaborn.pydata.org/generated/seaborn.barplot.html>

⁴³ <https://seaborn.pydata.org/generated/seaborn.swarmplot.html>

4.7.2.global_analysis

This is the main function that allows us to study the interactions based on our chosen study variable.

The function is called with the following way:

global_analysis(df, time_df, study_columns, title, level, context_index, software, detail)

with the arguments representing, respectfully:

- **df**: this is the pandas DataFrame of the study angle we have chosen (iterations, modification phases or unique values).
- **time_df**: this is the corresponding DataFrame representing the evolution of the study variable over time.
- **study_columns**: this represents a list of the significant values based on the chosen study angle that we wish to compare between the various groups. Currently we can choose time, rate, total, maximum or coefficient of variation (CV). Other statistical values such as mean, skewness or standard deviation could easily be implemented, given the small sample size of this study this was not developed fully, no doubt with larger sample sizes a more robust statistical analysis would be pertinent at this step.
- **title**: this simply allows us to label our graphic.
- **level**: this is the variable with which we will group our data (software proficiency, order of interaction or context of interaction).
- **context_index**: this represents the context in which we want to perform our analysis (0,1 or 2) or None if we chose to perform an analysis across all three contexts.
- **software**: if we chose to perform our analysis based on a given software proficiency, this is where we can specify which software.
- **detail**: this is a Boolean value which indicates whether we would like to have the detail for each subject in each group or if we would like to group the results and only represent the average.

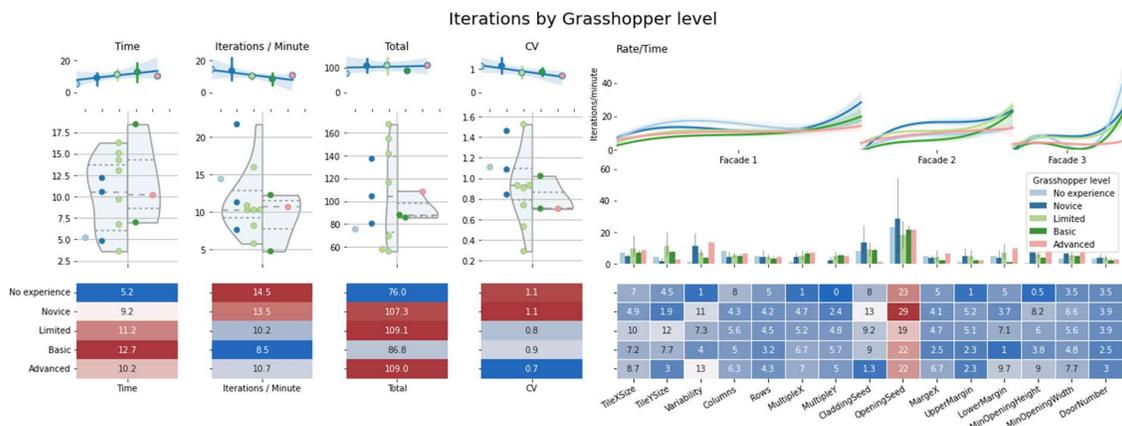


Figure 4.7-2 : global_analysis(iterations_df, time_iterations_df, ['Time', 'Rate', 'Total', 'CV'], 'Iterations', 'Level', None, 'Grasshopper', False)

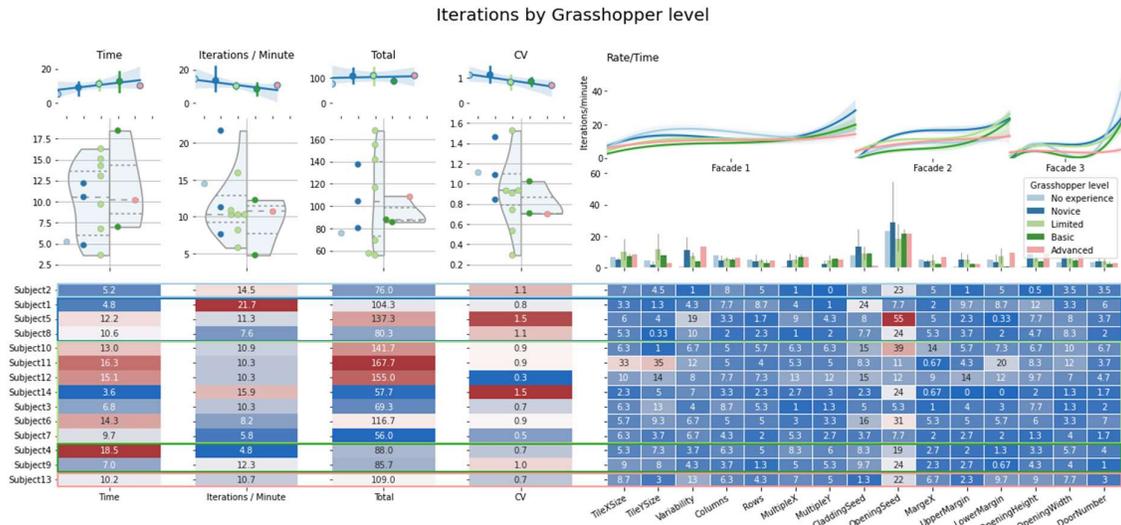


Figure 4.7-3 : `global_analysis(iterations_df, time_iterations_df, ['Time','Rate','Total','CV'], 'Iterations', 'Level', None, 'Grasshopper', True)`

The output gives us a dashboard like that on Figure 4.7-2 or Figure 4.7-3 (depending on whether we wish a detailed dashboard or not), and can be split in three categories:

On the left (Figure 4.7-4) we can find four columns, one for each of the significant values we chose, with, from top to bottom:

- A **linear regression plot**⁴⁴ across the different groups, to study the evolution from group to group and see if there is a global trend. On this plot is also represented the mean for each group along with an error bar corresponding to the confidence interval.
- A **violin plot**⁴⁵ of the values showing the discrete data points. This violin plot is split when studying by software proficiency, with the left representing the three lower proficiency levels and the right the three higher proficiency levels, to better exacerbate the difference between less and more experienced subjects.
- A **heatmap**⁴⁶ showing the mean value for each of the groups and colored from blue to red depending on where on the range of values they fall (blue being the lowest, red the highest, and the middle represented by white).

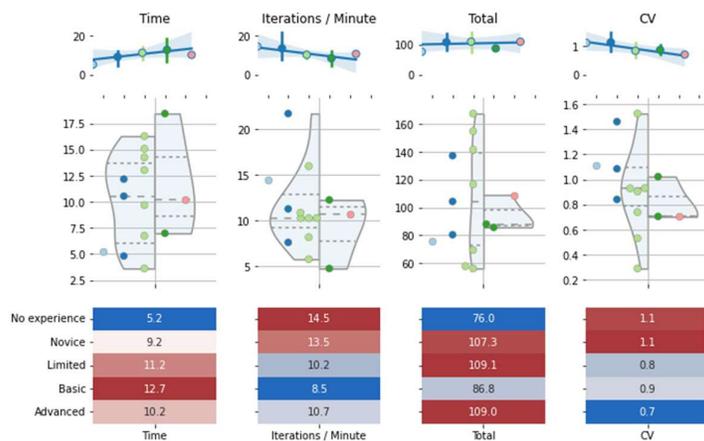


Figure 4.7-4 : Example of significant value graphic analysis

⁴⁴ Implemented using seaborn (<https://seaborn.pydata.org/generated/seaborn.regplot.html>)

⁴⁵ Implemented using seaborn (<https://seaborn.pydata.org/generated/seaborn.violinplot.html>)

⁴⁶ <https://seaborn.pydata.org/generated/seaborn.heatmap.html>

On the top right (Figure 4.7-5), is plotted the evolution of the rate of the study angle during the experiments, approximated by a cubic polynomial regression plot⁴⁷ with 10 data points per façade (as explained in section 4.5), with one curve for each of the groups and the interactions of the façades clearly separated. The rates are normalized according to the time each subject spent on each façade, with the relative widths of the sections for each of the façades proportional to the average proportion of time spent on each of them by the subjects.



Figure 4.7-5 : Rate evolution

On the bottom right (Figure 4.7-6), we can see a combination of a bar plot with the confidence intervals represented through error bars and a heatmap for the mean values pertaining to each of the input parameters that the subjects interacted with. This is also where the legend for the dashboard appears, with a coherent color code across the different dashboards based on the different groupings that are possible.

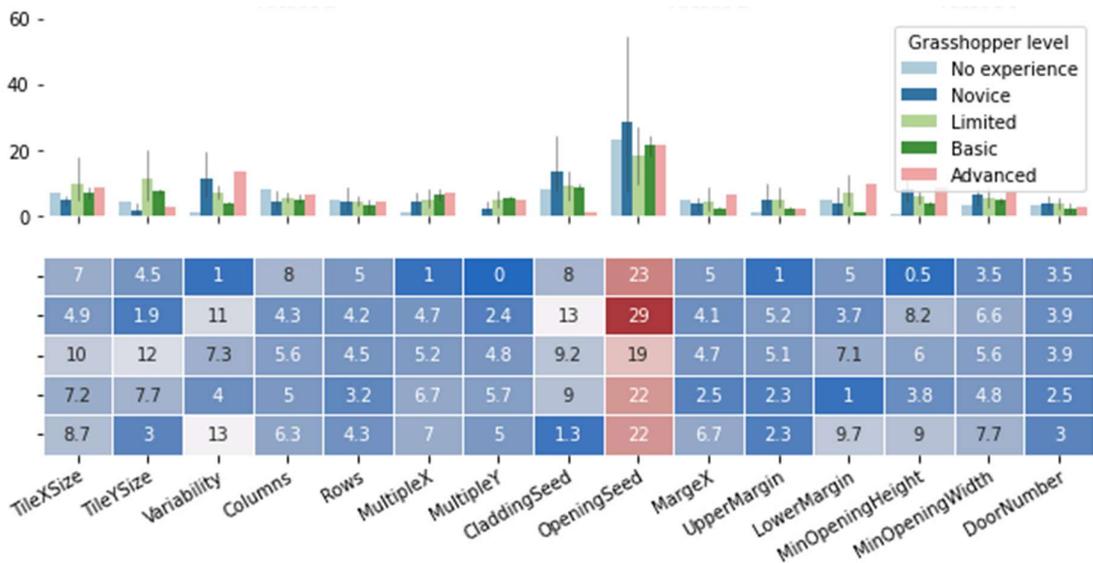


Figure 4.7-6 : Parameter values

These data visualizations were produced for every experimentation. The most significant ones are used to illustrate the research results, the others are available either in the Appendix or online on the GitHub⁴⁸ repository.

⁴⁷ <https://seaborn.pydata.org/generated/seaborn.regplot.html>

⁴⁸ <https://github.com/XGar/Impact-of-context-familiarity-on-computational-design-logic-appropriation>

5. Data analysis

The harvested data are first analyzed to assess the validity of the experience, especially:

- The global behavior across the different contexts, in order to identify possible flaws in the experimentation set up (5.2)
- The impact of the order in which the subjects interacted with the different implementations (5.3)
- The impact of previous software knowledge (5.4).

Then, the data are analyzed to answer our research question: how context familiarity impacts the user behavior (5.5).

5.1. Harvested data

5.1.1. Successful experimentations and collected data points

A total of 14 experiments were conducted, among both current students in the architectural engineering section of Liège University and recent graduates of the section, 7 and 7 respectively.

As discussed previously (section 4.4), during each interaction, data was automatically recorded every time the user changed a parameter. In total, 92990 points of data for the interactions were collected.

During the first 3 experiments, the automated data harvesting for the Hybrid interaction failed. Using the backup screen recordings (explained in section 4.3.1), this data was harvested manually, except for Subject2 who was unluckily also the only subject for which the screen recording also failed. Because of that, Subject2 was disregarded for the analysis in the Hybrid context. It should also be note that for some of the interactions in the Hybrid context, subjects at time suffered significant lag which was detrimental to their fluid use of the tool.

This raw data was cleaned and processed as seen in sections 4.5 and 4.6. A total of 328 graphs were generated; amongst them 148 were considered relevant and grouped in categories of four to form the 37 appendixes. The others can be found on the GitHub repository of the project.

5.1.2. Characterization of the population

Software proficiency in AutoCAD and SketchUp was relatively high for all the subjects, whereas knowledge in Rhino and Grasshopper is more representative of average proficiency in the different software (Figure 5.1-4). Only two of the 14 subjects had experience with Blender or 3dsMax, therefore these were discarded from the analysis.

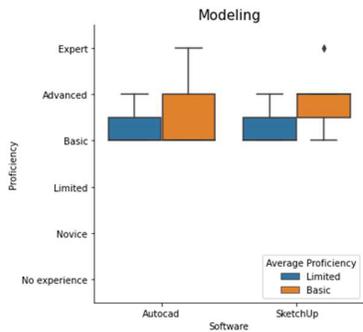


Figure 5.1-1 : Modeling proficiency distribution

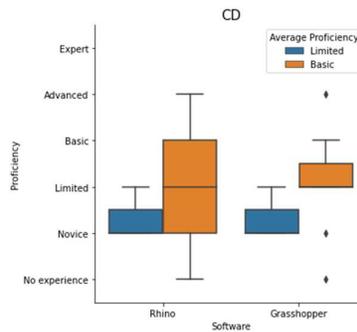


Figure 5.1-2 : CD proficiency distribution

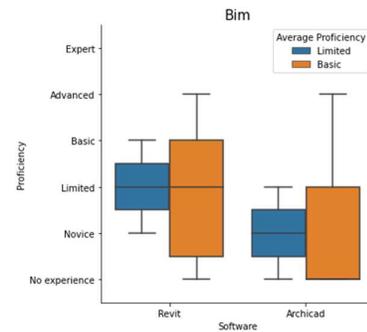


Figure 5.1-3 : BIM proficiency distribution

Subject1	3	3	1	1	2	0	3	2	1
Subject2	4	4	0	0	0	1	4	1	0
Subject3	4	4	2	2	3	0	4	3	2
Subject4	3	3	3	3	4	0	3	4	3
Subject5	4	4	1	1	3	1	4	3	1
Subject6	3	3	2	2	1	2	3	2	2
Subject7	3	4	4	2	0	0	4	0	3
Subject8	4	4	1	1	3	0	4	3	1
Subject9	3	4	3	3	2	1	4	2	3
Subject10	5	5	1	2	2	0	5	2	2
Subject11	3	4	2	2	3	3	4	3	2
Subject12	3	3	1	2	0	4	3	4	2
Subject13	3	3	3	4	1	3	3	3	4
Subject14	4	4	2	2	3	0	4	3	2
	Autocad	SketchUp	Rhino	Grasshopper	Revit	Archicad	Modeling	Bim	Rhino+GH

Figure 5.1-4 : Software proficiency chart

5.2. Differences between different Contexts of interaction

The first thing we will assess is the overall behavior across the different contexts of interaction. The objective is to better understand the specificities of how differently the subjects interacted with the same underlying logic simply based on the context (as described in section 4.1). This will allow us to better interpret our results when we evaluate how previous software experience influences usage of a foreign logic in each context (section 0). To do so, we compared the characteristics of the interviewee as harvested in the surveys with information from the experimentations, namely the study of:

- the number of iterations (every time a parameter was changed)
- the modification phases (when the user switched between parameters)
- the number of unique values (unique parameter values explored)

5.2.1. Survey

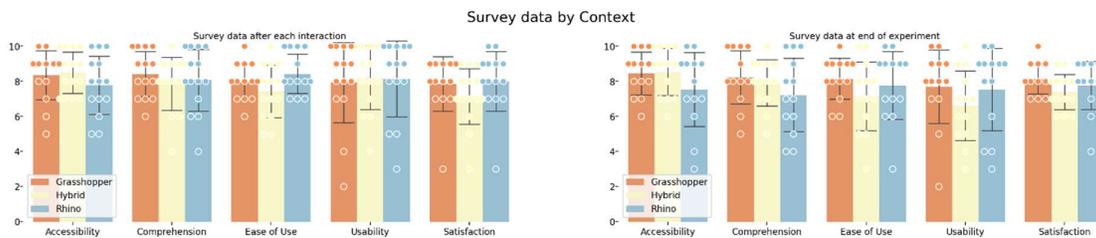


Figure 5.2-1 : Survey data by context of interaction

The survey data (Figure 5.2-1), reveals the following on the perception of the experimentation by the interviewee:

- The **interaction in Rhino was consistently perceived as slightly less accessible** from the start while both the grasshopper and hybrid implementations ranked similarly.
- The **three implementations were initially perceived as similar in terms of comprehension** by the subjects, **while the Rhino implementation was noted slightly lower in the final survey.**
- The **hybrid implementation was consistently ranked as being less easy to use.** This can be explained by the difficulties and low fluidity caused by the unoptimized implementation, specifically the contextual sliders which at times induced lag. Subjects found the **Rhino and Grasshopper implementations equally easy to use**, with Rhino being perceived as slightly easier to use in the first survey while the opposite was expressed in the final survey.
- **The subjects' level of satisfaction at being able to achieve their desired outcome, showed trivial difference in the answers before and after the experimentations**, apart from the fact that the second survey showed lower variance. The implementations in **Grasshopper and Rhino gave subjects equal satisfaction** while they were on average **less satisfied in the Hybrid implementation.** This could again be in part explained by the technical issues which arose in this implementation, particularly given the fact that not all subjects noted it poorly. We can also see that two subjects noted the Rhino implementation less favorably, which they justified by the limitation in their capacity to edit the logic, specifically the bounds of values that can be given to the rows and columns parameters.
- When asked how they could see themselves using tools and logic implemented in similar ways based on the context, while **initially all three contexts performed similarly**, **in the final survey subjects rated Grasshopper as the highest, followed closely by Rhino then by the Hybrid version which did more poorly.**

5.2.2. Number of iterations

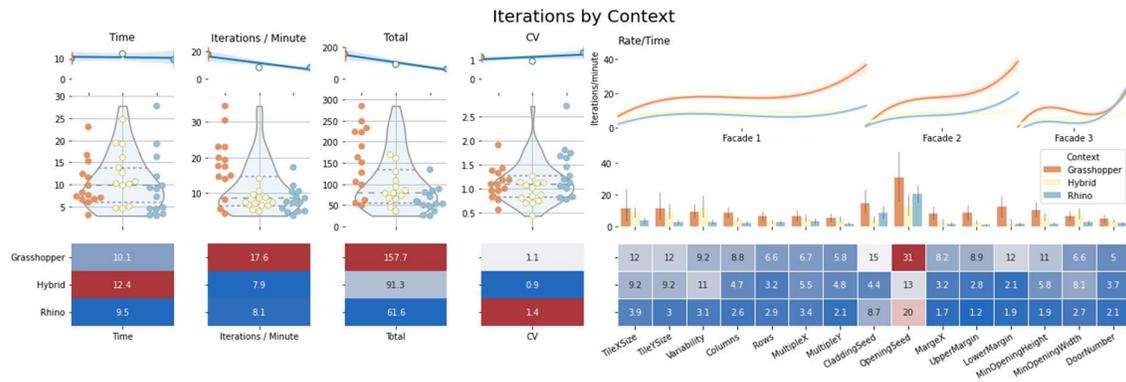


Figure 5.2-2 : Iterations by Context

Concerning the number of iterations made by the subjects as seen in Figure 5.2-2, we can see immediately that:

- The subjects spent a **similar amount of time between the contexts**, but this amount of **time varies greatly** (5 to 25 minutes) in each context.
- The **Grasshopper rate of interaction is around twice as high** as in the other two contexts, on average. However, a closer look at how that rate evolves over the interaction reveals that this is particularly the case at the beginning but **by the third façade the rate of interaction is more similar across the different contexts**. This would suggest that a study conducted over a longer period might deliver different results and that we could hypothesize that the subject gets used to the design logic and **begins using it rather than simply exploring it**.
- The **total number of iterations varies more in Grasshopper** than in the other contexts.

5.2.3. Number of modification phases

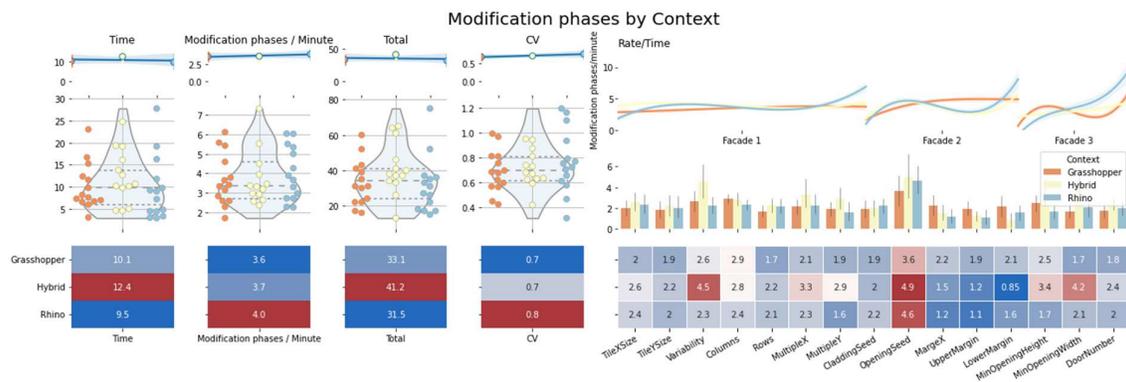


Figure 5.2-3 : Modification phases by Context

By studying the subjects' behavior in terms of count of modification phases and how it evolves over the interaction (Figure 5.2-3), we can see that the subjects **changed which parameters they were interacting with at a similar rate across all three implementations**, slightly more when interacting in Rhino, and observing the behavior over the interaction duration seems to indicate that subjects became **progressively more at ease switching between parameters in the Rhino implementation**.

5.2.4. Unique Values

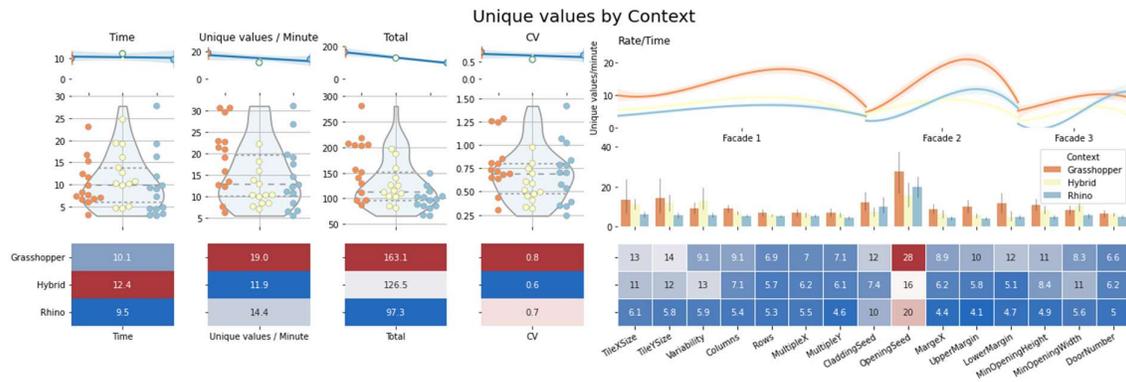


Figure 5.2-4 : Unique values by Context

Studying the evolution of unique values depending on the context of interaction reveals that **subjects explored a wider range of unique solutions overall when interacting through grasshopper and at an average higher rate**, however this difference is more clearly marked during the first 2 façades, with the average rate during the last façade stabilizing to be closer to that of the other 2 contexts.

5.2.5. Summary

The overall study of the variance in interaction behavior and logic perception across the different implementations reveals that while there are some differences (namely the higher rate of interaction in Grasshopper), **these differences for the most part seem to stabilize over time** or can be explained by things specific to the implementations (such as the fluidity problems in the hybrid implementation or the bounds of accepted values in the Rhino implementation) more so than the context. While there are differences in the interactions from subject to subject, the way the design logic was implemented in the different contexts seems to be at best only partly responsible for these differences.

5.3. Order of interaction impacted behavior

In this section, we will study how the order in which the interactions were done had an impact on the behavior and perception of the subjects. We will first look at the general results, and if necessary, will discuss how these results vary across the different implementations.

5.3.1. Survey

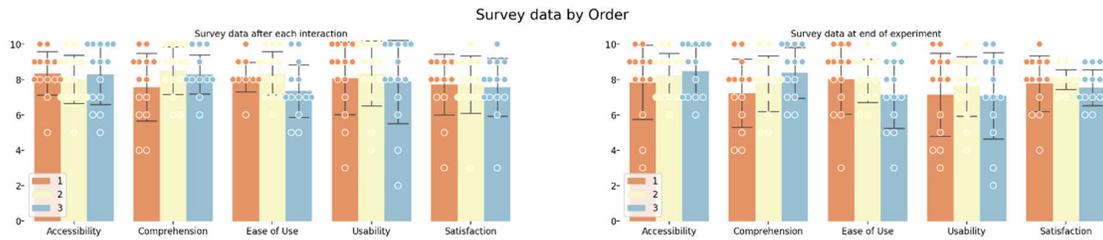


Figure 5.3-1: Survey data by Order

The average survey results shown on Figure 5.3-1 reveal **the order of interaction had an impact on the subjects' perception**. While immediately following each interaction the average perceived accessibility was similar across the 3 different interactions, **when evaluating them at the end of the experiment the subjects seem to rank the interactions as more and more accessible**. This could be explained by the fact that the subjects became progressively more familiar with the exercise. This hypothesis is strengthened by the fact that the perceived level of comprehension of the underlying logic also increased for every interaction, both immediately following and at the end of the experiment. Another notable result we can see is that **the last interaction was consistently perceived as being less easy to use**.

Studying the impact of the order on the different implementations (Appendix 19-1, Appendix 28-1, Appendix 37-1) shows that these general observations do not reflect across all three contexts.

In Grasshopper there is no clear progression of the level of accessibility with each successive interaction, and the level of comprehension is perceived as less favorable and decreases with each interaction, as do the other categories of the survey, especially for "Usability". This indicates that **while Grasshopper was well appreciated when it was the first method of interacting with the logic, when users had already used another implementation, they tended to be harsher when evaluating it during the final survey**.

In the **Hybrid implementation**, the general trend of increasing level of comprehension can be seen, while **ease of use, usability and satisfaction tended to decrease with each successive interaction**.

In **Rhino** however, **the results increased across the board for each successive interaction**, especially in the final survey. This seems to indicate that in this implementation, getting familiar with the logic took some time (longer than in Grasshopper for example) but once familiar with the exercise the subjects appreciated it. The difference between the progression of this implementation versus the others also explains why the global results are perhaps not representative.

5.3.2. Number of iterations

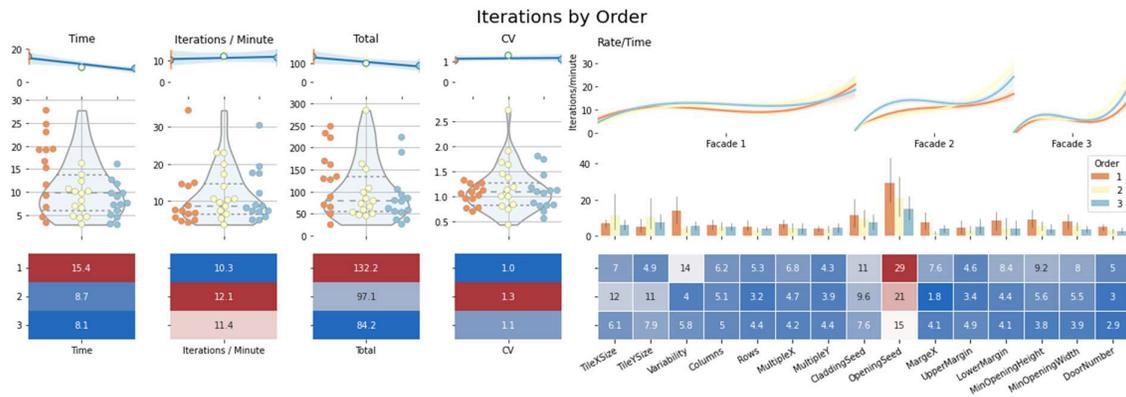


Figure 5.3-2 : Iterations by Order of interaction

The only notable observation we can identify by studying the average number of iterations by order of interaction (Figure 5.3-2) is that **on average, subjects spent twice as long on their first interaction as the following two.**

A more in depth look in each context (Appendix 19-2, Appendix 28-2, Appendix 37-2), shows that **this is the case in each context of interaction.**

5.3.3. Number of Modification Phases

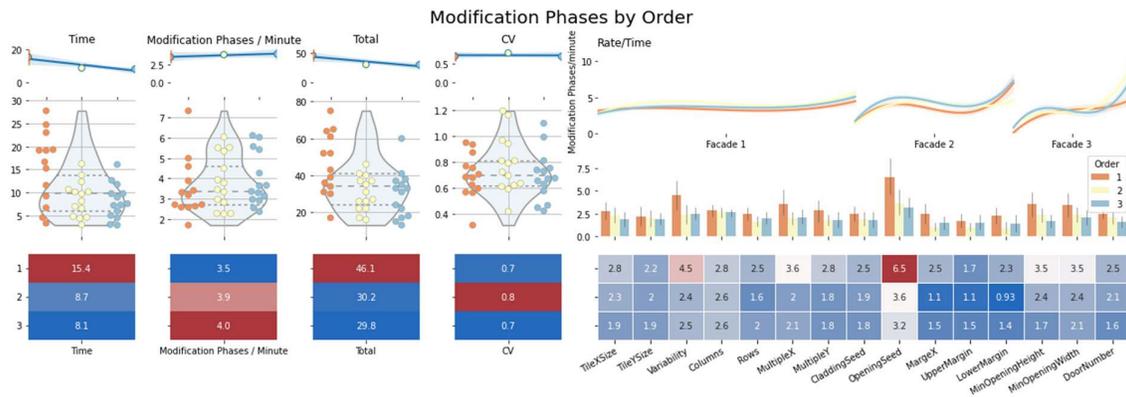
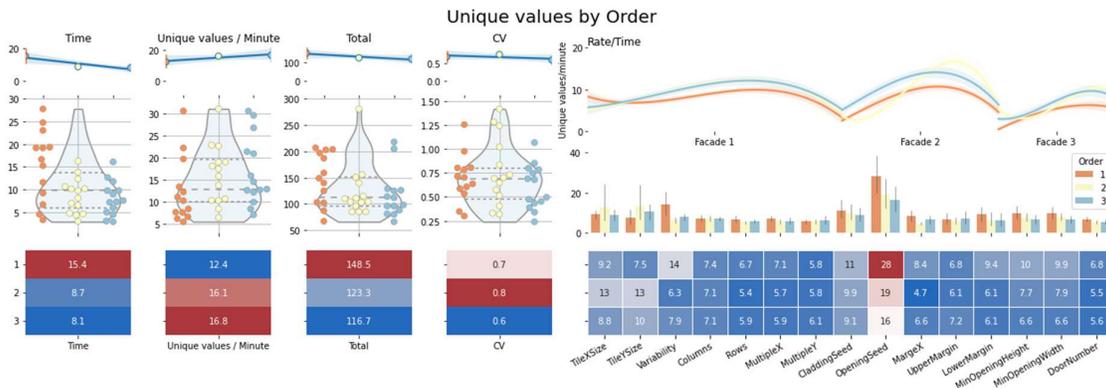


Figure 5.3-3 : Modification phases by Order of interaction

Studying the global evolution of the modifications phases depending on the order of the interaction (Figure 5.3-3) shows a tendency of **slightly increasing rate for each successive interaction.** This global trend is also found in Grasshopper (Appendix 19-3) and Rhino (Appendix 37-3) while the opposite seems to be true in the Hybrid implementation (Appendix 28-3).

5.3.4. Number of unique values



The study of unique values, like that of the modification phases, shows a **tendency of increasing rate** both globally and in each implementation (Appendix 19-4, Appendix 28-4, Appendix 37-4), but this difference seems to be **primarily between the first interaction and the following two**. This could be explained by the fact that as stated previously, subjects spent around twice as long on their first interaction as the following 2, while observing the total number of unique values of the parameters explored shows they only explored slightly more in that amount of time. Observing the evolution of the rate over the experiment further shows that for each façade, **the rate of new unique values plateaus then decreases after a certain amount of time**.

5.3.5. Summary

From this analysis, we can conclude that **the order in which the subjects interacted with the design logic in the different contexts had a pronounced impact in how they interacted with it**, especially for the **first interaction where they discovered the underlying design logic**, whereas for the **following two they only discovered how it was implemented**. We shall have to take this into account for the rest of our analysis and what conclusions we can make from them, especially given the small sample size in relation to the diversity of the profiles of the subjects in regard to their proficiency in the various software.

5.4. Previous software experience impacted the interaction

In this section we will study if and how previous experience in different software had an impact on the interactions. To do so we will refer to the generated Dashboards that can be found in the appendix.

5.4.1. AutoCAD

Survey data

Surprisingly, **experienced users of AutoCAD seemed to have slightly more favorable views on the interactions** (Appendix 1-1), particularly when it came to the ease of use and day to day usability of such a logic, as well as a better feeling of comprehension of the underlying logic. The link between the rating of **ease of use and usability was less clear cut when looking at the Grasshopper implementation** (Appendix 11-1), however in that implementation a similar **link between AutoCAD experience and level of comprehension** can be observed. For the **Hybrid implementation** (Appendix 20-1) **no clear relationship** could be seen, considering the low sample size (1) in the “Expert” category. In **Rhino, a relation trend between proficiency in AutoCAD and the survey results** can be seen across all categories (although sometimes in only one of the two surveys), with higher proficiency levels correlating with more favorable survey answers (Appendix 29-1).

Number of iterations

Studying the number of iterations values at a global level based on AutoCAD experience yields **no clear observations** (Appendix 1-2), nor did that experience have a discernable impact for those metrics in Grasshopper (Appendix 11-2), the Hybrid context (Appendix 20-2), or in Rhino (Appendix 29-2).

Modification phases

This was also the case for the study of the modification phases (Appendix 1-3, Appendix 11-3, Appendix 20-3, Appendix 29-3).

Unique values

This was also the case for the study of the unique values explored (Appendix 1-4, Appendix 11-4, Appendix 20-4, Appendix 29-4).

Summary

When analyzing the harvested data by grouping users by AutoCAD proficiency level, while the **survey data did show a link on average and in Rhino especially**, the three implemented **study angles for characterizing the tool interactions revealed nothing**. However, this could very well be due to our low sample size.

5.4.2. SketchUp

Survey data

Observing the subjects' average survey results (Appendix 2-1) seems to indicate that the categories for **Ease of use and Usability are generally rated higher by more experienced SketchUp users**. The other categories also show slight increases by SketchUp level but not enough to draw definitive conclusions. **In Grasshopper, this general trend is echoed** for the Usability category in the 2nd survey, but much less so for Ease of Use (Appendix 12-1). Another interesting observation is that while the level of comprehension seemed to be directly correlated with a subject's level in SketchUp in the survey conducted immediately after the interaction, this is much less the case in the survey conducted at the end of the three interactions. **No clear conclusions could be drawn for the Hybrid context** based the survey data when cross referencing it with the subjects' level in SketchUp (Appendix 21-1). Once again, the interactions in **Rhino showed the greatest correlation between higher SketchUp levels and positive survey answers**, across all categories (Appendix 30-1), although this could be a product of the small sample size of the "Expert" category.

Number of iterations

Studying the number of iterations shows an interesting difference between the behavior in the different contexts. **In Grasshopper, the rate of iterations seemed to decrease with higher levels of proficiency** (Appendix 12-2), while the **opposite tendency can be observed in Rhino** (Appendix 30-2). The global study (Appendix 2-2) and the one for the Hybrid context (Appendix 21-2) showed no clear relationship between Sketchup level and behavior.

Modification phases

This is **also the case** when we look more in depth at the evolution of the modification phases, albeit to a **lower extent**. (Appendix 2-3, Appendix 12-3, Appendix 21-3, Appendix 30-3)

Unique values

Finally, when observing the results of the different analysis of the unique values generated, we can see that there is **no discernable impact** of one's level in SketchUp in general or in any of the contexts. (Appendix 2-4, Appendix 12-4, Appendix 21-4, Appendix 30-4)

Summary

Although experienced SketchUp users seemed to prefer their interactions in Rhino, and indeed interacted more quickly there than in Grasshopper, they did not explore more solutions or achieve their results faster.

5.4.3. Rhino

Survey data

Observing the global survey cross referenced with Rhino level reveals **no clear trends** of a relationship (Appendix 3-1). Furthermore and perhaps surprisingly, while the survey results varied significantly from level to level in the different contexts (Appendix 13-1, Appendix 22-1, Appendix 31-1), **no clear relationship between a subject's level and their evaluation of the different categories** of the survey could be observed, **not even in Rhino** where we could have expected that more experienced Rhino users would have had a bias towards giving better evaluations of the different categories. **This reveals that while someone might have experience in 3d modeling, even in a familiar context, the use of computational design logic also requires experience in computational thinking** (Carpo, 2017) (de Boissieu, 2022).

Number of iterations

On average, more **experienced users of Rhino seem to interact with the logic at a slower rate** but interacted with the logic a similar amount of time on average as less experienced users (Appendix 3-2). This global trend can also be seen in the interactions which took place in Grasshopper (Appendix 13-2) although more experienced users tended to interact with the logic for a longer amount of time, **in the Hybrid context this trend was more pronounced** (Appendix 22-2), **and most significantly in Rhino** (Appendix 31-2).

Modification phases

Studying the way in which users changed from parameter to another across all contexts shows that on average there is not a significant difference between users of different Rhino levels (Appendix 3-), nor is there in the Hybrid context (Appendix 22-3). In Grasshopper and Rhino however, there seems to be a slight tendency for users with a higher level to **switch less frequently between parameters**, this tendency is more marked in Grasshopper (Appendix 13-3) but in Rhino we can also observe that more experienced users interacted with the different parameters more evenly rather than mainly focusing on switching between a couple of parameters, which results in a lower coefficient of variation between the number of modification phases by parameter (Appendix 31-3).

Unique values

Finally, studying the average number of unique values of the different parameters traversed by the users indicates that **more experienced Rhino users tested fewer unique values** per minute and tested unique values of the different parameters **much more evenly on average**

Appendix 3-4). This was **also the case in Grasshopper** (Appendix 13-4) **and Rhino** (Appendix 31-4) with the additional trend of more **experienced users interacting with the logic for longer in Grasshopper**. In the **Hybrid context, there was no clear relationship** between a user's level in Rhino and how they explored unique values of the different parameters (Appendix 22-4).

Summary

Based on the harvested data, it would appear that more experienced Rhino users did not benefit from their experience when it came to interacting with the computational design logic, nor were their perceptions different than those of the other subject groups, as can be seen by the survey data.

5.4.4. Grasshopper

Survey data

The average survey data by Grasshopper level (Appendix 14-1) shows **no clear relationship** between subjects' grasshopper levels and their evaluation of any of the categories of the survey. This is slightly biased by the fact that some levels contain only one subject. Removing these levels or grouping them with the others still does not reveal clear relationships though.

Looking more closely at the data for the survey specific to the Grasshopper context we can make the following observations:

- In the survey immediately following the interaction, more experienced users had a tendency of rating their level of comprehension of the underlying logic lower than less experienced users but were more likely to see themselves using computational design logic in this way day to day.
- In the survey conducted at the end of the experiment, **more experienced users seemed to rate the accessibility and ease of use of the implementation as higher** than less experienced users.

In the Hybrid context (Appendix 23-1), the second survey data shows a **clear relationship between higher levels in Grasshopper and better levels of comprehension**. The other categories showed elevated levels of variance from level to level, but no clear link is apparent.

In Rhino (Appendix 32-1), the survey data does not lend itself to any definitive conclusions, in part due to the low sample size of the 2 levels on the extremes. Concentrating only on the experience levels in which there are more than one subject, we could observe that the perception of usability decreases with grasshopper level.

Number of Iterations

Looking at the average iterations by Grasshopper level (Appendix 4-2), slight negative correlations between a user's level in grasshopper and the rate of interaction as well as the coefficient of variation of the average number of iterations of the different parameters can be observed, as well as a positive correlation with the rate of interaction. This is also the case in the Grasshopper and Rhino implementations (Appendix 14-2, Appendix 32-2) particularly for the time spent on the interaction, while the Hybrid context implementation shows much less correlation between these metrics and a user's Grasshopper level (Appendix 23-2).

Modification phases

The rates of the average modification phases (Appendix 4-3) are also inversely proportional to the users' level in grasshopper, to a similar proportion as for the number of iterations, but the CV no longer follows that trend. Looking more closely at the different contexts, we can see that this global trend is mostly due to the Rhino implementation (Appendix 32-3) rather than the other 2 (Appendix 14-3, Appendix 23-3).

Unique values

The unique values followed the trends shown for the number of iterations, as much for the average values (Appendix 4-4) as for the different contexts, with Grasshopper (Appendix 14-4) and Rhino (Appendix 32-4) showing more sensitivity to a user's level in grasshopper than in the Hybrid implementation (Appendix 23-4).

5.4.5. Archicad

Survey data

The average survey data by Archicad level is not easily interpreted, some trends can be seen in the surveys conducted immediately after the interactions like an increasing perception of usability with higher levels in Archicad as well as greater levels of satisfaction with their ability to achieve their desired results. Looking at the data collected in the survey conducted at the end of the experiment does not reveal the same trends, however (Appendix 5-1). Similar trends and ambiguities can be found in the data of the 2 surveys conducted for the Grasshopper context by Archicad level (Appendix 15-1) preventing us from making any definitive conclusions. The surveys for the other 2 contexts cross referenced with Archicad level are also inconclusive (Appendix 24-1, Appendix 33-1).

Number of Iterations

Studying the average number of iterations by Archicad level (Appendix 5-2), we can see that more experienced users on average spent more time doing the interactions and interacted more evenly with the different parameters, but at a similar rate as the users with a lower Archicad level. This trend is not reflected in Grasshopper (Appendix 15-2), instead more experienced users interacted at a higher rate. In the Hybrid implementation (Appendix 24-2), there is variance in the values by level of the different metrics for the number of iterations (rate, total, cv) such that no clear relationship between those metrics and Archicad experience can be established. Furthermore, looking at how the rate of iteration evolves over time for the different level groups shows that there are significant differences in how different people interact with the logic. In Rhino (Appendix 33-2), we can observe that more experienced users tend to interact at a slower rate and more evenly between the different parameters compared to less experienced users.

Modification phases

Observing the average number of modification phases by Archicad level overall (Appendix 5-3), we can see that the rate is relatively stable (ranging from 3.1 to 3.9 modification phases / minute on average), with the most variation occurring for the lowest level of Archicad experience. We can also see a decreasing CV with increasing levels in Archicad, and that while behavior in terms of the evolution of the rate over time is similar between levels across the first 2 façades, by the 3rd one there are bigger differences between the different level groups. For the interactions taking place in the Grasshopper (Appendix 15-3) and Hybrid (Appendix 24-3) contexts, there is no apparent relationship between the way users went from parameter to parameter and their level in Archicad. In the Rhino implementation however, we can see that more experienced users interacted with a lower average rate and lower CV than less experienced users. (Appendix 33-3)

Unique values

Globally (Appendix 5-4), no clear trend could be observed, nor could any definitive links be seen in the Grasshopper (Appendix 15-4) or Hybrid implementations (Appendix 24-4). In Rhino (Appendix 33-4), we can observe that the rate decreases with Archicad level as well as the CV between parameters.

5.4.6.Revit

Survey data

The average survey data by Revit level (Appendix 6-1) seems to indicate that **as the level of Revit proficiency increased in users, their rating of the accessibility, comprehension and ease of use categories was susceptible to decrease**. Further investigation in the three contexts reveals no clear trends in any of them (Appendix 16-1, Appendix 25-1, Appendix 34-1).

Number of Iterations

Studying the global behavior in terms of the number of iterations by Revit level (Appendix 6-2) reveals **no trends** apart from a seemingly slight tendency for more experienced users to interact less evenly between the different parameters and focus primarily on a couple few. **No trends could be observed in Grasshopper either** (Appendix 16-2), while **in the Hybrid context more proficient Revit users spent longer on average** than less proficient users (Appendix 25-2), and how the rate of iterations evolved over time varies between the distinct levels. In Rhino (Appendix 34-2), the only observation we can make is that while for the first façade the rates of interaction varied in the same range between different levels, for the second and third façades the range of rates of interaction varies more, with higher levels in Revit reaching higher rates, particularly towards the end of their conception of the façade.

Modification phases

The study of the modification phases revealed **no definitive trends**, either on average (Appendix 6-3), or in any of the contexts (Appendix 16-3, Appendix 25-3, Appendix 34-3).

Unique values

- Global (Appendix 6-4): More experienced users concentrated more on certain parameters in their exploration of unique values (higher CV).
- Grasshopper (Appendix 16-4): Less exploration of unique values by more experienced Revit users (lower total).
- Hybrid (Appendix 25-4): More unique values explored by more experienced users, but only on certain parameters (higher total and CV).
- Rhino (Appendix 34-4): No clear differences between subject groups.

5.4.7. BIM

Survey data

- Global (Appendix 7-1): Lower Accessibility, Comprehension and Ease of Use rating by more experienced BIM users.
- Grasshopper (Appendix 17-1): Higher Usability perceptions for more experienced BIM users.
- Hybrid (Appendix 26-1): High variability between subject groups and between surveys so no definitive conclusions can be made.
- Rhino (Appendix 35-1): No conclusions can be made.

Number of Iterations

- Global (Appendix 7-2): No notable trends apart from the fact that more experienced users spent longer at a similar rate on their interactions.
- Grasshopper (Appendix 17-2): no clear trends.
- Hybrid (Appendix 26-2): More time spent at a higher rate of interaction for more experienced users.
- Rhino (Appendix 35-2): The most experienced users spent the most time but at the lowest rate of interaction and the most evenly across the different parameters.

Modification phases

- Global (Appendix 7-3): no new trends could be seen.
- Grasshopper (Appendix 17-3): slightly lower rate for more experienced users.
- Hybrid (Appendix 26-3): More time spent but at the same rate for more experienced users.
- Rhino (Appendix 35-3): same behavior as observed when studying the number of iterations.

Unique values

- Global (Appendix 7-4): no new trends could be seen (same as number of iterations and modification phases).
- Grasshopper (Appendix 17-4): no clear trends.
- Hybrid (Appendix 26-4): More time spent but at the same rate for more experienced users.
- Rhino (Appendix 35-4): The most experienced users spent the most time but at the lowest rate of new unique values and the most evenly across the different parameters.

5.4.8. Modeling

Survey data

- Global (Appendix 8-1): minimal differences between subject groups, slight positive link between subjects' levels and their perception of comprehension and, for the survey conducted at the end of the experiment, the ease of use and day to day usability of the context of implementation.
- Grasshopper (Appendix 18-1): same observations as for the global study.
- Hybrid (Appendix 27-1): no clear trends can be observed.
- Rhino (Appendix 36-1): variability between the 2 surveys, but general correlation between higher proficiency levels and more favorable answers across all the categories.

Number of Iterations

- Global (Appendix 8-2): No discernable difference in interaction behavior between the different subject groups.
- Grasshopper (Appendix 18-2): no significant differences between subject groups.
- Hybrid (Appendix 27-2): no significant differences between subject groups.
- Rhino (Appendix 36-2): Higher rates of interaction for more experienced users.

Modification phases

- Global (Appendix 8-3): lower average total number of modification phases for increasing proficiency levels.
- Grasshopper (Appendix 18-3): lower average rates for more experienced subject groups, but this especially the case due to the behavior towards the end of the context interaction (end of façade 2 and façade 3) whereas at the beginning of the context interaction (façade 1) the rates of change between parameters are similar across subject groups.
- Hybrid (Appendix 27-3): similar behavior across subject groups.
- Rhino (Appendix 36-3): no clear trends except less time spent on the interaction on average by more experienced subjects.

Unique values

- Global (Appendix 8-4): No significant differences between subject groups.
- Grasshopper (Appendix 18-4): No definitive trends.
- Hybrid (Appendix 27-4): No significant differences between subject groups.
- Rhino (Appendix 36-4): No significant differences between subject groups.

5.5. Context familiarity impacted behavior

5.5.1. Grasshopper

In this section, we will investigate whether more experienced grasshopper users truly benefited from their experience when it came to using a previously unknown script not made by them. While this analysis has already been done to some extent in the previous section, we will provide a more in-depth analysis and hypothesis as to the reasons of some of the observations.

Survey data

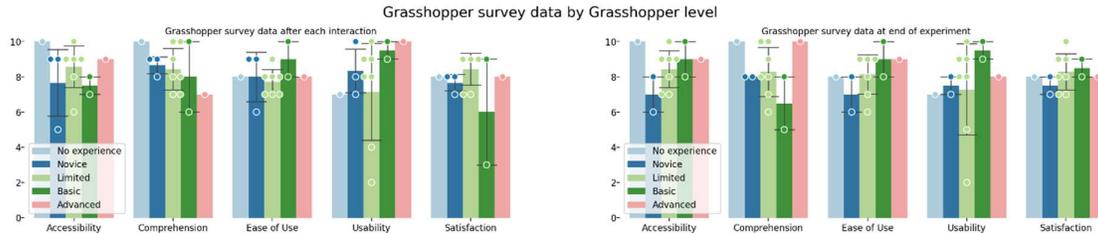


Figure 5.5-1: Grasshopper survey data by Grasshopper level

Observing the feedback answers (Figure 5.5-1) given by the subjects grouped by their stated Grasshopper level, we can observe the following:

- While immediately following the interaction in grasshopper no clear link between a subjects stated perception of the accessibility of the implementation and their experience level in grasshopper could be seen, when looking at the survey answers given at the end of the experiment, it seems that **the more experienced a subject is in grasshopper, the more accessible this method of interacting with the design logic seemed to them**, which is to be expected.
- On the other hand, **more experienced grasshopper users stated having more difficulty understanding the underlying design logic**, with however a notable difference in the answers for the 2 surveys or the lone subject in the “Advanced” experience group. This could be explained partly by the fact that they are more used to creating their own design logic, and partly because the logic was by choice not purely parametric, which they are more used to.
- While initially following the interaction in Grasshopper the different subject groups rated the ease of use of the implementation similarly, at the end of the experiment a clear trend where **more experienced users find the implementation easier to use** can be seen.
- Concerning the day-to-day usability of general computational design implemented in this way, unsurprisingly, the **more experienced users found this method of interaction more usable day-to-day**.
- **Different subject groups rated their level of satisfaction at achieving their desired outcomes similarly.**

From this data we can conclude that while subjects with more experience in grasshopper were influenced by that experience and more at ease, lack of experience did not affect less experienced subjects’ ability to use the design logic and achieve their desired outcomes.

Number of iterations

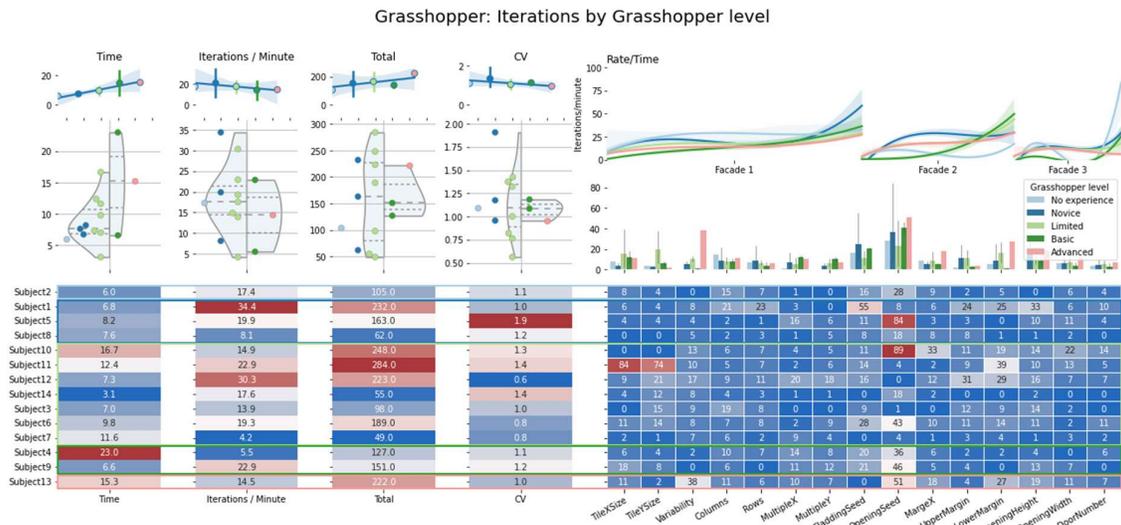


Figure 5.5-2 : Detailed study of the number of iterations in Grasshopper, by Grasshopper level

From the detailed study of the number of iterations in Grasshopper, by Grasshopper level (Figure 5.5-2) we can observe the following:

- Subjects more familiar with grasshopper spent on average more time on the interaction.
- More experienced subjects did not interact with the logic at a higher rate, even at a slightly lower rate in fact.
- Their behavior was more constant within the subject groups.
- The rate of interaction of more experienced users did not evolve in a significantly different manner than less experienced subjects.

Based on this study variable and the harvested data, there is no indication that more experienced subjects in grasshopper interacted significantly differently than novices or even subjects with no prior experience in Grasshopper.

Number of modification phases

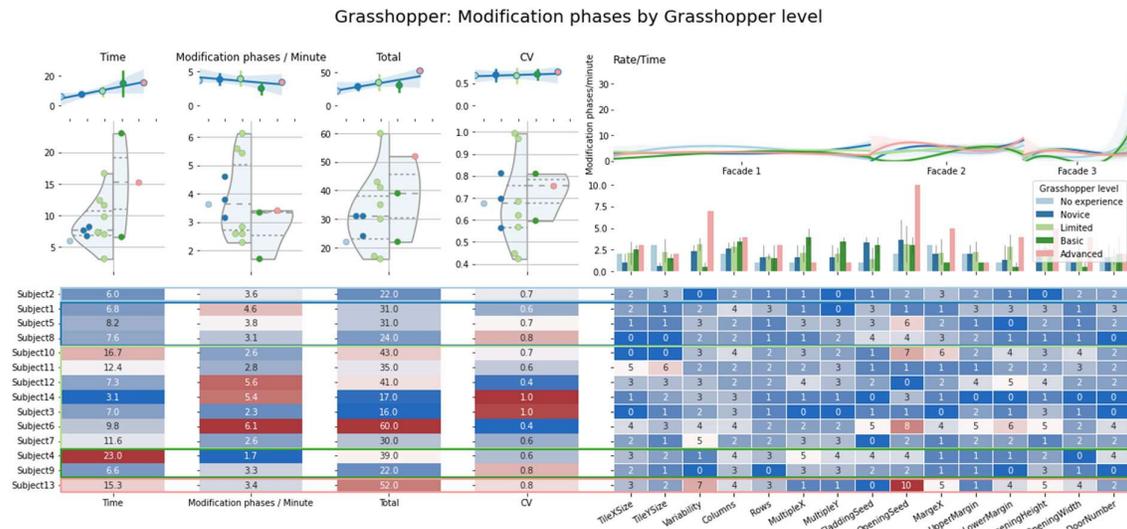


Figure 5.5-3 : Detailed study of the number of modification phases in Grasshopper, by Grasshopper level

From the detailed study of the number of modification phases in Grasshopper, by Grasshopper level (Figure 5.5-3) we can only observe the following:

- The rate of change between parameters was slightly lower on average for more experienced users, although this could be explained by the fact that they spent more time interacting in this context.

Based on this study variable and the harvested data, as in the previous observations, there is no indication that more experienced subjects in grasshopper interacted significantly differently than novices or even subjects with no prior experience in Grasshopper.

Number of unique values

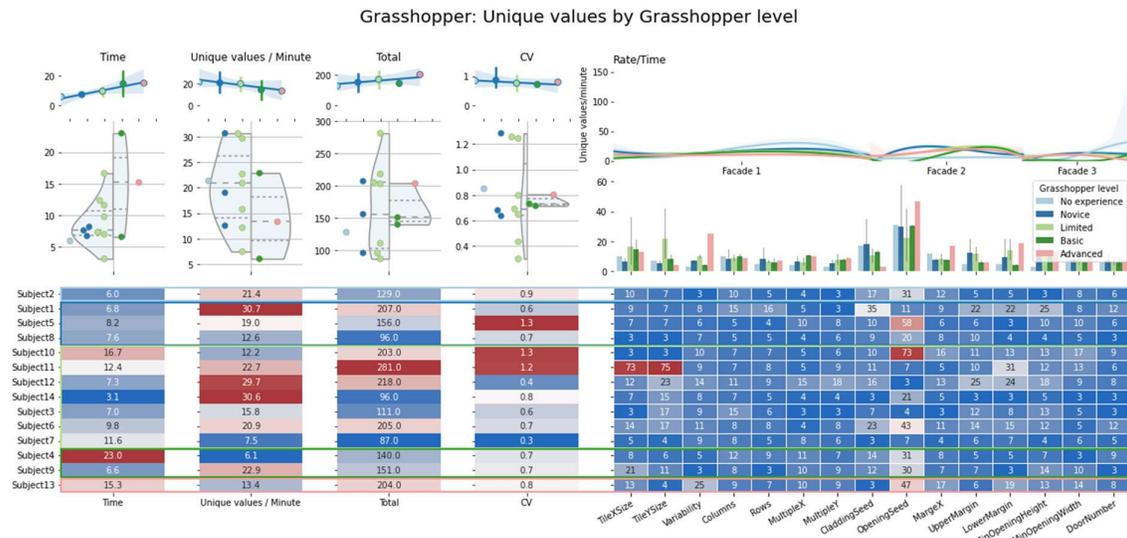


Figure 5.5-4 : Detailed study of the number of unique values in Grasshopper, by Grasshopper level

From the detailed study of the number of modification phases in Grasshopper, by Grasshopper level (Figure 5.5-3) we can only observe the following:

- While more experienced subjects explored a greater number of unique values, they did so at a lower average rate, and more evenly across the different parameters.
- Observing the link between time spent and the rate seems to indicate a decreasing rate after a certain period of time spent, which we can also see by observing the evolution of the rate over the interaction, where towards the end of a façade a decrease was seen in the number of new unique values as the user converged toward their preferred solution.

As in previous observations, there is no indication that more experienced subjects in grasshopper interacted significantly differently than novices or even subjects with no prior experience in Grasshopper.

Summary

In conclusion, based on the harvested data and chosen study parameters, there seems to be **no negative impact on a subject's use of computational design logic in grasshopper because of their inexperience or unfamiliarity with the context** that can be described as significant or sustained.

5.5.2. Hybrid

The study of the impact of context familiarity on the Hybrid implementation is not as easy to do given the fact that this implementation used a custom UI that none of the subjects could have been previously exposed to. In a way, this simulates the situations found when using new unfamiliar tools (such as web-based platforms discussed in section **Error! Reference source not found.**). However, given the fact that the interaction takes place in the Rhino viewport, and parameters are interacted with either through manipulating points in the view or through sliders, this implementation is a hybrid between the Rhino and Grasshopper contexts. For that reason, the hypothesis was taken that users most familiar with both Rhino **and** Grasshopper could be considered as most “familiar” with this context. This is why we will study this context by grouping the subjects by their average experience in Rhino and Grasshopper, as explained on page 29 .

Survey data

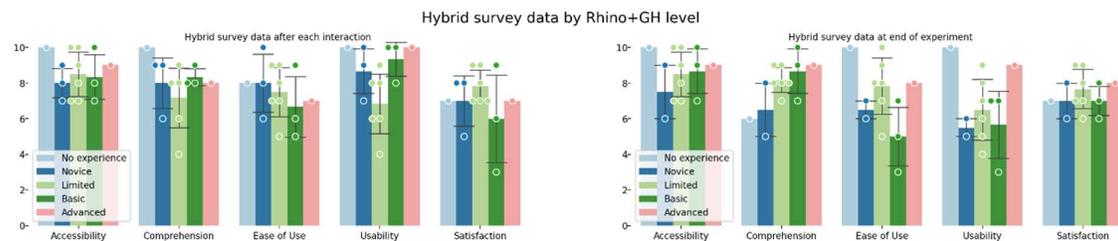


Figure 5.5-5 : Hybrid survey data by average Rhino-GH level

Studying the survey data for the Hybrid context based on subject average level in Rhino and Grasshopper reveals:

- The only clear trend that can be observed is that in the survey conducted at the end of the experiment, **more experienced users rated their ability to understand the design logic in this implementation as higher than less experienced users.**
- Of all the categories, the **level of satisfaction is the most consistent between the surveys and most homogenous within and among the different subject groups.**
- While there are differences between the subject groups, the differences in the 2 surveys as well as the small sample size and the impact of the order makes drawing any other conclusions impossible.

From this data we can see that while the answers vary greatly from person to person, **the link between these answers and previous experience in Rhino and Grasshopper is far from clear** and looking at the survey data grouped by the various other software as can be seen in the appendix (20-27) reveals a **link in no other software or category such as BIM or modelling** either. The **most definitive impact on the answers is in fact the order** in which the subjects did this implementation as can be seen in Section 5.2 and Appendix 28-1.

Furthermore, given that independently of these variations from subject to subject **they all appeared similarly satisfied with their ability to achieve their desired outcome** (particularly as stated at the end of the experiment), this lends further credence to the hypothesis that in practice the context of interaction and a user’s previous experience with it have no or a limited impact on their ability to effectively use a computational design logic, especially given the relatively poor implementation in this case comparatively to the other 2 and the problems that arose from it as explained in section 5.1.1.

Number of iterations

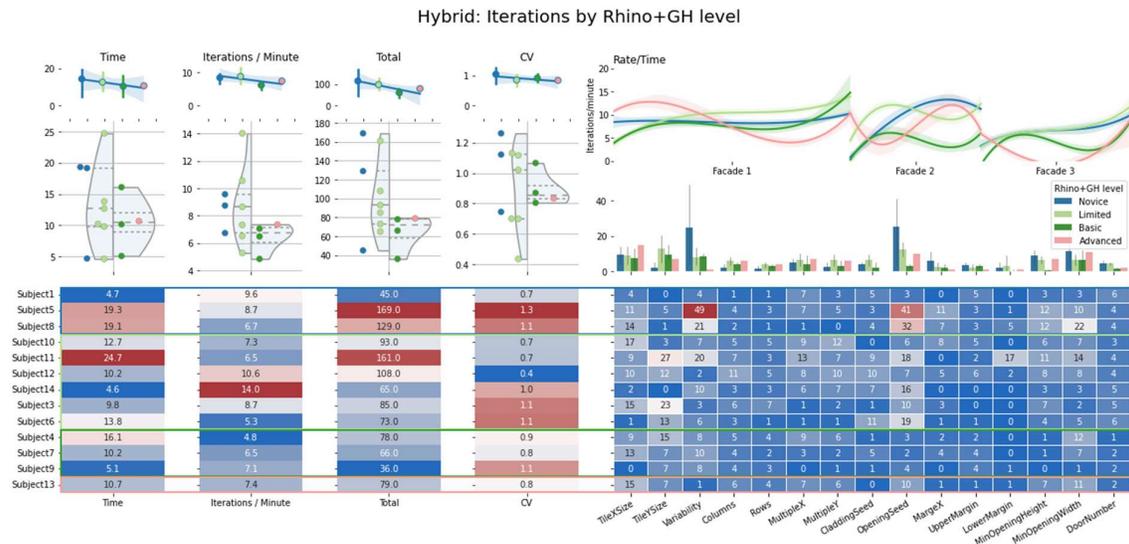


Figure 5.5-6 : Detailed study of the number of iterations in the Hybrid context, by average Rhino and Grasshopper level

From the detailed study of the number of iterations in the Hybrid context, based on subjects’ average proficiency in both Rhino and Grasshopper (Figure 5.5-6), the following observations can be made:

- More experienced subjects spent less time interacting in this context compared to less experienced subjects.
- More experienced subjects also interacted with the design logic at a lower average rate, although this rate of interaction occasionally spiked and matched or even surpassed that of lower-level subject groups. This indicates that in this implementation, the rate of interaction was not constant across the duration of the interaction in the context.
- The highest spikes in the number of iterations across the parameters can be seen in the parameters used to explore variations of the established conditions and the average numbers were substantially higher in the Novice subject group compared to the higher-level subject groups.

Observing the study of the number of iterations in this context based on proficiency levels in the different software (Appendix 20 to Appendix 27) reveals that Rhino and Grasshopper were indeed the software where the biggest differences could be seen between subject groups of different software proficiency levels, but also that differences could be seen based on subjects BIM proficiency (Appendix 26-2), but with opposite trends as those seen here.

Number of modification phases

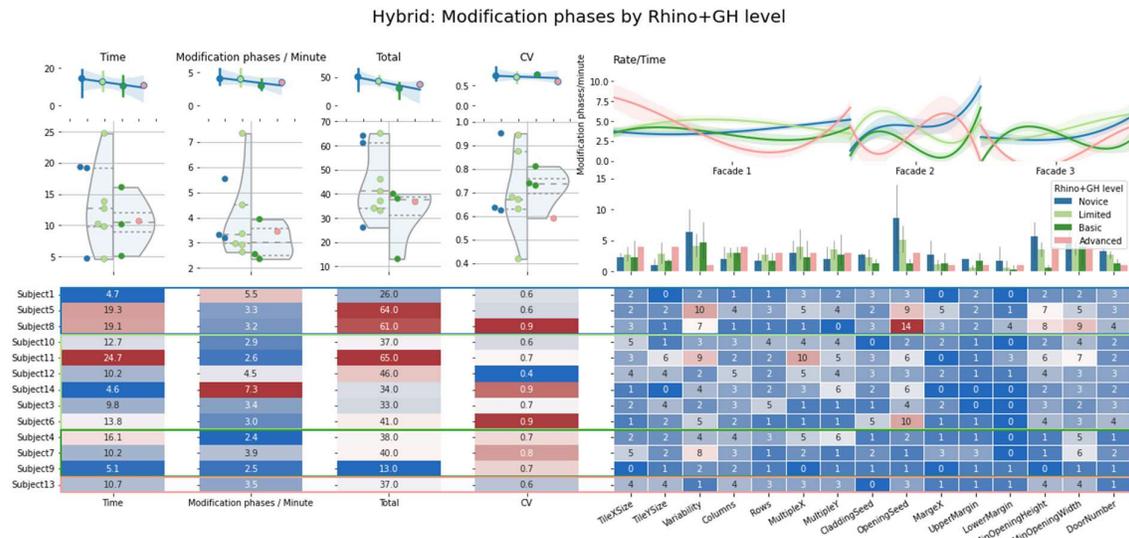


Figure 5.5-7 : Detailed study of the number of modification phases in the Hybrid context, by average Rhino and Grasshopper level

The detailed study of the number of modification phases in the Hybrid context, by average Rhino and Grasshopper proficiency, (Figure 5.5-7) reveals no observations different than those already discussed in the previous section. Observing the evolution of the rate over time shows just how variable the behavior of interaction from subject to subject was in this implementation.

Comparing to the different analysis based on proficiency level in the other software, we can again see that Rhino and Grasshopper are indeed the software in which a subjects proficiency level impacts their interaction in this context of implementation the most, and that although BIM proficiency also seems to be correlated based on the harvested data, its impact has the opposite effect, with more experienced users changing between parameters at a higher rate.

Number of unique values

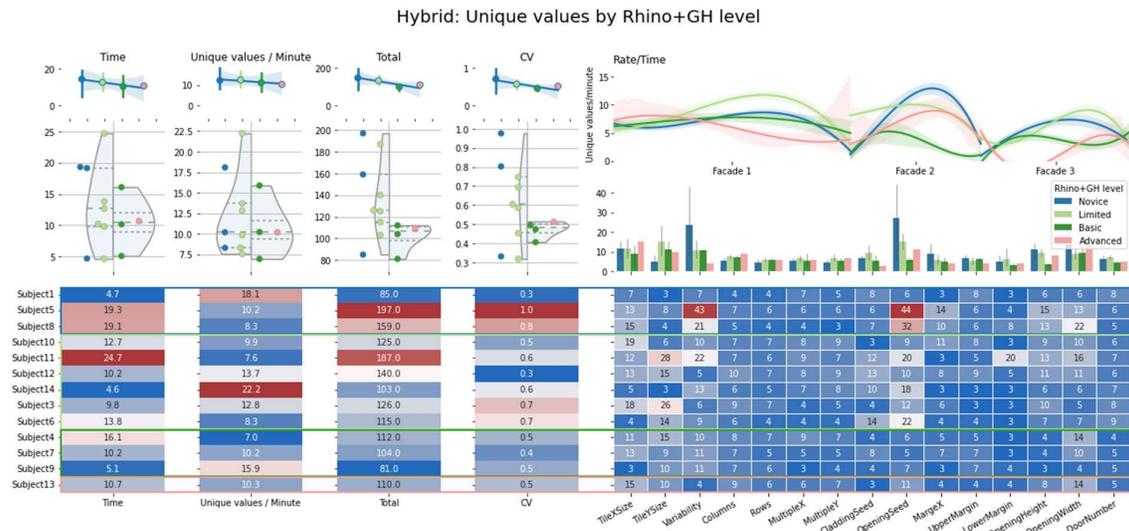


Figure 5.5-8 : Detailed study of the number of unique values in the Hybrid context, by average Rhino and Grasshopper level

Looking at the detailed study of the number of unique values in the Hybrid context, by average Rhino and Grasshopper level, (Figure 5.5-8) the same observations can be made, albeit to a lesser extent. This can be explained by the fact that as we can see by observing the evolution of the rate over time, after a certain amount of time for each façade, the rate at which users explore new unique values decreases as they converge on their final solution which is usually composed of previously explored values for the different parameters. As such longer periods of time spent on the interaction do not have as great an impact on the various significant values we can observe in the dashboard.

5.5.3. Rhino

As in the previous 2 sections, we will investigate if subjects more experienced in Rhino truly benefitted from that experience when it came to interacting with a computational design logic within its context.

Survey data

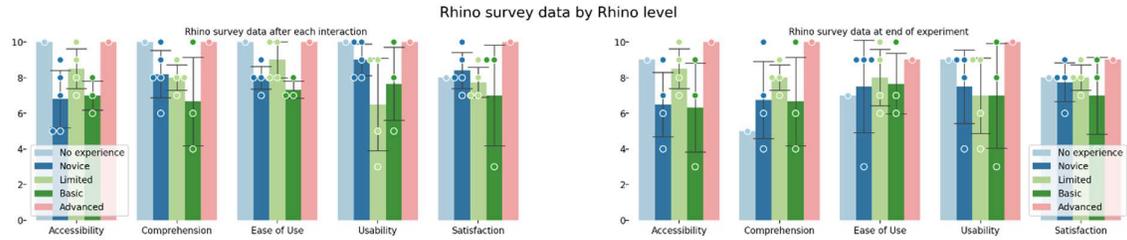


Figure 5.5-9 : Rhino survey data by Rhino level

Studying the survey data for the interactions done in Rhino, with subjects grouped by Rhino level, (Figure 5.5-9) we can see that while there are certainly large differences between the answers given by different subjects, grouping the subjects by their experience in Rhino reveals **no clear trend**. This might be partly explained by the impact the order in which the subjects interacted with this implementation had on their appreciation of it, as discussed in section 5.3 and can be seen in Appendix 37-1.

Comparing this observation to the survey data grouped by proficiency levels in the other software, we can see that it is when grouping the subjects by average proficiency in “classic” modeling software (Appendix 36-1) that the greatest link can be found between their answers and their stated proficiency level, with a positive correlation across all the categories (especially for Ease of Use) although the differences are minimal so this could very well be attributed to the order in which the subjects did the interaction or other factors altogether, given the small sample size.

Number of iterations

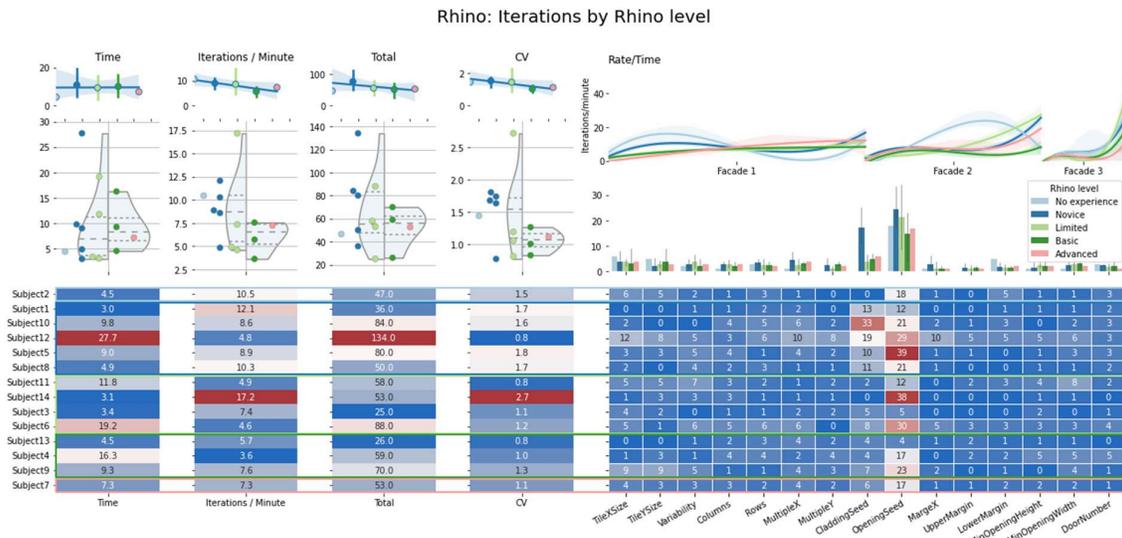


Figure 5.5-10 : Detailed study of the number of iterations in Rhino, by Rhino level

From the detailed study of the number of iterations in Rhino, grouping subjects by their stated proficiency in Rhino, (Figure 5.5-10), we can see that:

- For a similar amount of time spent on the interaction, **more experienced subjects interacted with the logic at a lower average rate.**
- **More experienced subjects also interacted with the different parameters more evenly,** although this observation is highly influenced by outliers in the lower-level subject groups and may be a product of the small sample size.
- Across all subject groups, **the rate of interaction of interaction gradually increased and peaked towards the end of use.**

From this study based on our harvested data, we can conclude that while experience in Rhino did impact the behavior of interaction, it was not in a way that was detrimental to less experienced subjects. Furthermore, the differences between subject groups seem to be mainly attributed to the higher use in less experienced subject groups of parameters that generated variations of solutions based on the established conditions, and this higher use was concentrated towards the end of the use of the logic for a given design problem.

We can then affirm that by looking at this study variable and our harvested data, **lack of familiarity with this interaction context is not an obstacle to the effective use of computational design logic.**

Modification Phases

Rhino: Modification phases by Rhino level

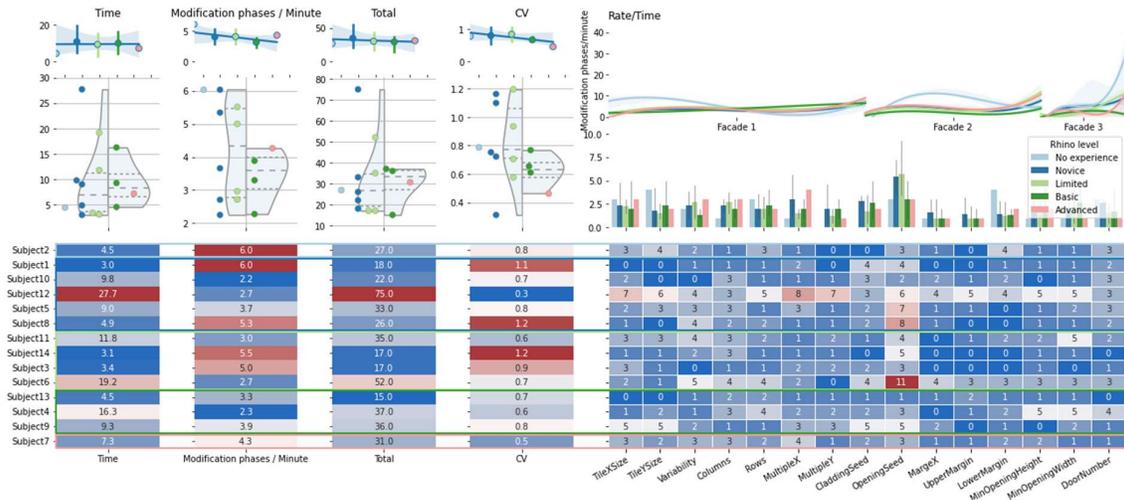


Figure 5.5-11 : Detailed study of the number of modification phases in Rhino, by Rhino level

The detailed study of the number of modification phases in Rhino, based on Rhino level, (Figure 5.5-11) reveals much of the **same observations as in the study of the number of iterations** done previously, albeit with less pronounced differences between subject groups, especially when it comes to how evenly the subjects switched between all the different parameters.

From this study variable and our harvested data, we find **no indication that lack of experience in this context is an obstacle to the use of computational design logic within it.**

Unique values

Rhino: Unique values by Rhino level

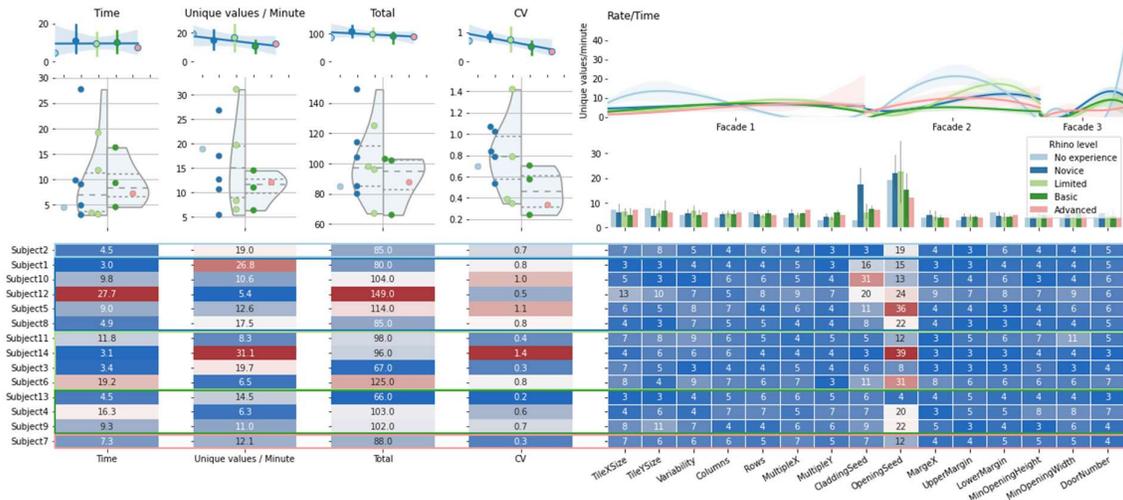


Figure 5.5-12 : Detailed study of the number of unique values in Rhino, by Rhino level

The detailed study of the number of unique values in Rhino, by Rhino level (Figure 5.5-12) reveals only a slight tendency that the more experienced users have of exploring unique values across the parameters more evenly. We can also see differences between subject groups in how the rate evolves during the exercise, although these differences are not echoed in the average values of the rates.

6. Results and discussions

By looking at the different results across the several ways the data was analyzed, even given our limited sample size and other factors prohibiting us from making definitive conclusions, we can at least state the following:

1. Across the test subjects and independently from the order in which they interacted with the different implementations, it seems that **one's familiarity with a given context of interaction mattered less than one's previous experience with computational design logic**, as could notably be observed in the users who only had experience with Rhino.
2. While previous familiarity with a context of implementation or computational design tended to lead to quicker initial interaction with the logic, **over the course of the experiment the differences between different skill or familiarity levels became less and less pronounced** with there often being no discernible difference that could be attributed to that factor towards the end of the interaction.
3. Independently of skill level, familiarity or order of interaction, the **interactions in Grasshopper where consistently rated higher and saw higher values** across the three study variables. This shows the impact that the user interface can have, although it should be noted that while a given user interface may lead to higher interaction rates in the discovery phase of a computational design logic and may help users to better understand it, this same user interface may not be ideal when it comes to actual production use of the same logic, as expressed by several of the test subjects. A longer experiment with a clear given objective on which to use a computational design logic would explore that hypothesis.
4. The difference in behavior between **a subject's first and second interaction varied significantly**, with subjects spending on average twice the amount of time on the first interaction where they discovered the computational design logic. While varying the order in which the subjects did the interactions mitigates this issue, by not disregarding this data or performing another type of analysis, this further corrupts any conclusions we could hope to make, in addition to the issues due to the small sample size.

7. Conclusion

The aim of this study was to investigate and attempt to measure the impact that the familiarity with a context of interaction had on a subject's use of a computational design logic.

The experiment and consequent analysis focused on the way subjects interacted with the exposed parameters of the design logic, by harvesting a detailed log of the interactions, and then comparing the interactions by grouping users by software proficiency level or order of interaction.

The proposed study metrics based on the raw data were the number of iterations, the number of modification phases and number of unique values across the duration of the interactions. These metrics were proposed and devised by me, and while they did reveal differences between the subject groups and contexts of interaction, they were quite correlated with one another on average and revealed only slight nuances that given the small sample size cannot be considered significant.

An improvement could be made by conducting either longer experiments, or by substantially increasing the sample size and subject pool, ideally both, with the advantage that the tools created for this study were made with this in mind, so this could be done with little additional effort apart from conducting the experiments.

Another improvement could be made by also categorizing the parameters, for instance by grouping them between boundary conditions, global parameters and variation drivers which would allow a better interpretation of the differences between the implementations and the different results (i.e. are the higher number of iterations in grasshopper due to an increase in all these categories compared the other implementations or did the subjects simply generate a higher number of variations by playing with the sliders).

While this subject merits further research, the amount of effort necessary to conduct the amount and kind of experiments that this work attempted falls far beyond the scope of this work, which only aimed to test experiment protocols, especially concerning the data harvesting, cleaning, engineering and visualization tools that were created to conduct the analysis.

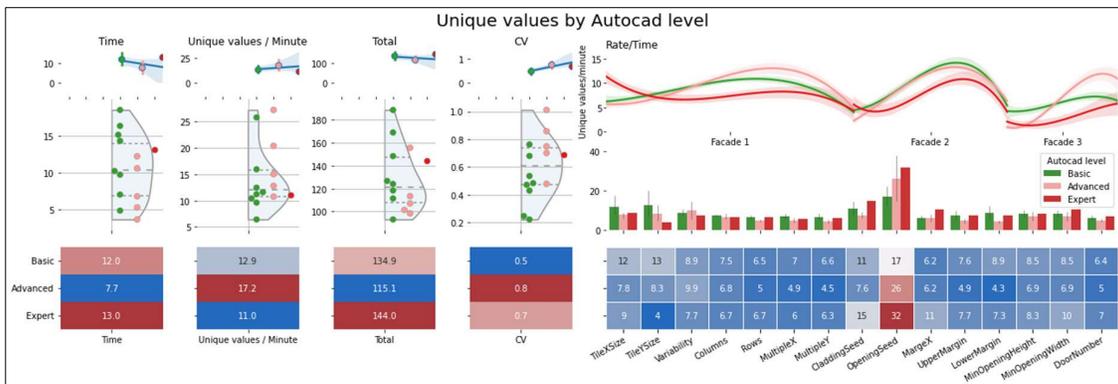
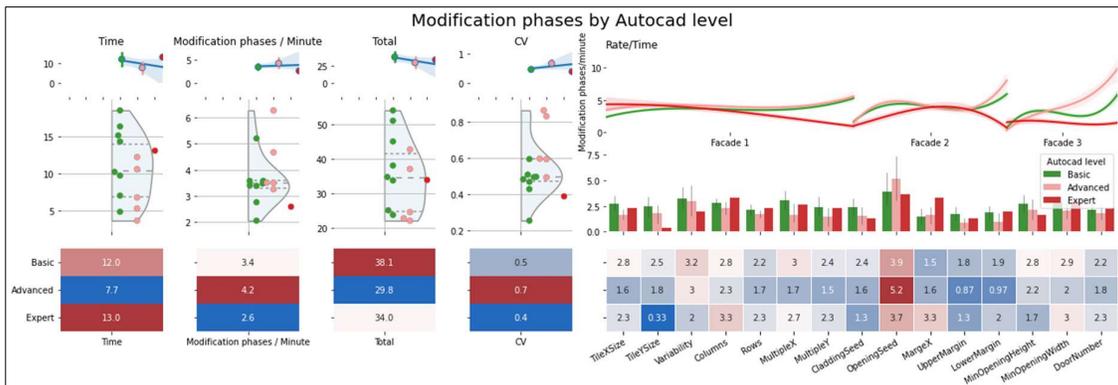
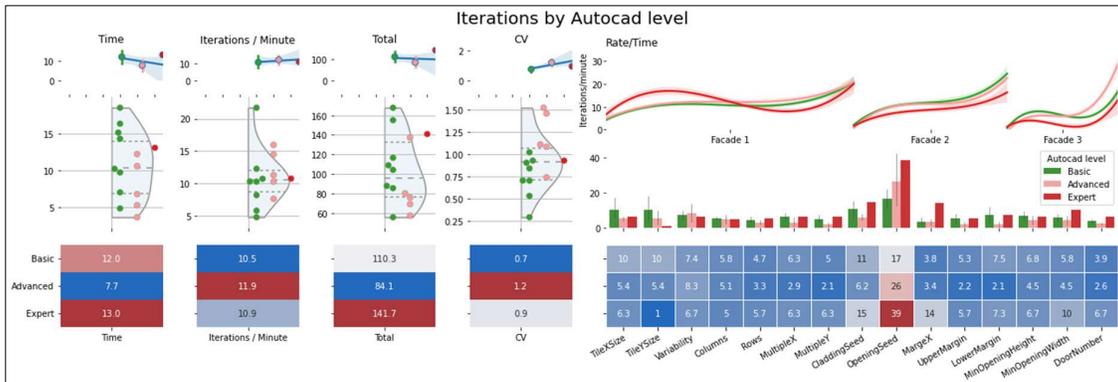
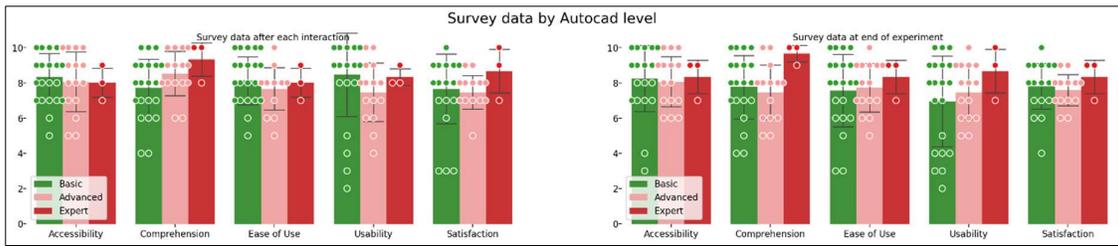
8. References

- Aish, R., & Bredella, N. (2017). The evolution of architectural computing: From Building Modelling to Design Computation. *Architectural Research Quarterly*.
- Caetano, I., Santos, L., & Leitão, A. (2020). Computational design in architecture: Defining parametric, generative, and algorithmic design.
- Calixte, X. (2021). *Les outils dans l'activité collective médiatisée en conception : traçabilité des usages au sein du processus de conception architecturale*.
- Calixte, X., Rajeb, B., & Leclercq, P. (2018). Traçabilité de l'usage des outils de conception dans un processus collaboratif. .
- Carpo, M. (2017). *The second digital turn: design beyond intelligence*. MIT Press.
- Charef, R., Emmitt, S., Alaka, H., & Fouchal, F. (2019). Building Information Modelling adoption in the European Union: An overview. *Journal of Building*.
- Cristie, V., Ibrahim, N., & Joyce, S. C. (2021). CAPTURING AND EVALUATING PARAMETRIC DESIGN EXPLORATION IN A COLLABORATIVE ENVIRONMENT: A study case of versioning for parametric design. *26th International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA)*. 2, pp. 131-140. Hong Kong: CAADRIA.
- Dautremont, C., Jancart, S., Dagnelie, C., & Stals, A. (2019). Parametric design and BIM, systemic tools for circular architecture.
- Davis, D. (2011). *Datamining Grasshopper*. Retrieved from <https://www.danieldavis.com/datamining-grasshopper/>
- Davis, D. (2013). Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture.
- Davis, D. (2020). Architects versus Autodesk. *ARCHITECT magazine, American Institute of Architects*.
- Davis, D. (2021). *CAD's Boring Future and Why it's Exciting*. Retrieved from <https://www.danieldavis.com/cads-boring-future/>
- de Boissieu, A. (2020). Superusers or super-specialists? Mapping the catalysts for the digital transformation of architectural practices.
- de Boissieu, A. (2021). Architecture et pratiques orientées sur les données : Pour un décloisonnement du BIM et du design computationnel. In *Anticrise architecturale, Analyse d'une discipline immergée dans un monde numérisé* (pp. 237-248). PUL Presses Universitaires de Louvain.
- de Boissieu, A. (2022). Introduction to Computational Design: Subsets, Challenges in Practice and Emerging Roles. In M. Bolpagni, R. Gavina, & D. Ribeiro, *Industry 4.0 for the Built Environment. Structural Integrity, vol 20* (pp. 55-75). Springer, Cham.
- Deutsch, R. (2019). *Superusers: design technology specialists and the future of practice*. Routledge.
- Fok, W., & Picon, A. (2016). *Digital property: open-source architecture*. Wiley.
- Gallas, M.-A., Jacquot, K., Jancart, S., & Delvaux, F. (2015). Parametric Modeling: An Advanced Design Process for Architectural Education.

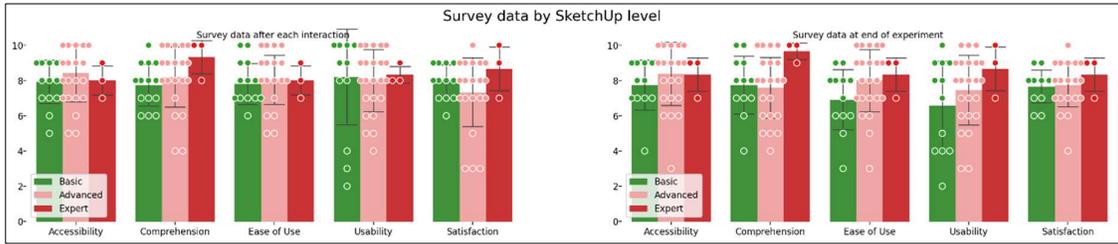
- Menges, A., & Ahlquist, S. (2011). *Computational Design Thinking*. Wiley.
- Otjacques, B. (2008). *Techniques de visualisation des informations associées à une plate-forme de coopération*.
- Oxman, R. (2017). Thinking difference: Theories and models of parametric design thinking.
- Peters, B., & De Kestelier, X. (2013). *Computation works: the building of algorithmic thought*. Wiley.
- Ribeirinho, M. J., Mischke, J., Strube, G., Sjödin, E., Blanco, J. L., Palter, R., . . . Andersson, T. (2020, June). The next normal in construction: How disruption is reshaping the world's largest ecosystem. *McKinsey & Company*.
- Stals, A. (2019). *Pratiques numériques émergentes en conception architecturale dans les bureaux de petite taille – Perceptions et usages de la modélisation paramétrique*. Thèse de doctorat, Université de Liège.
- Stals, A., Elsen, C., & Jancart, S. (2017). Practical Trajectories of Parametric Tools in Small and Medium Architectural Firms.
- Terzidis, K. (2006). *Algorithmic Architecture*. Architectural Press.
- Terzidis, K. (2011). Algorithmic Form. In A. Menges, & S. Ahlquist, *Computational Design Thinking*. Wiley.
- Van der Zee, A., & De Vries, B. (2008). Design by computation. *GENERATIVE ART CONFERENCE, 11*, pp. 35-37.
- Vrouwe, I., Dissaux, T., Jancart, S., & Stals, A. (2020). Concept Learning Through Parametric Design : A Learning Situation Design for Parametric Design in Architectural Studio Education. *Education and research in Computer Aided Architectural Design in Europe eCAADe20*. Berlin.

9. Appendix

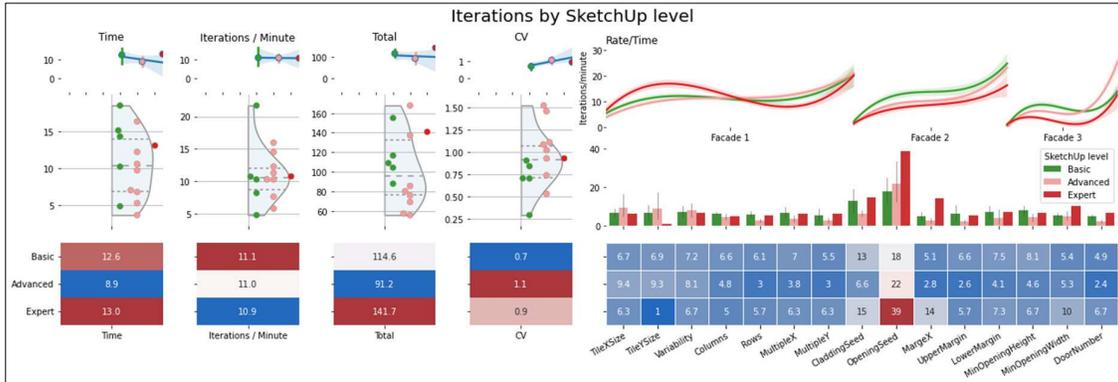
Appendix 1 : Average by AutoCAD



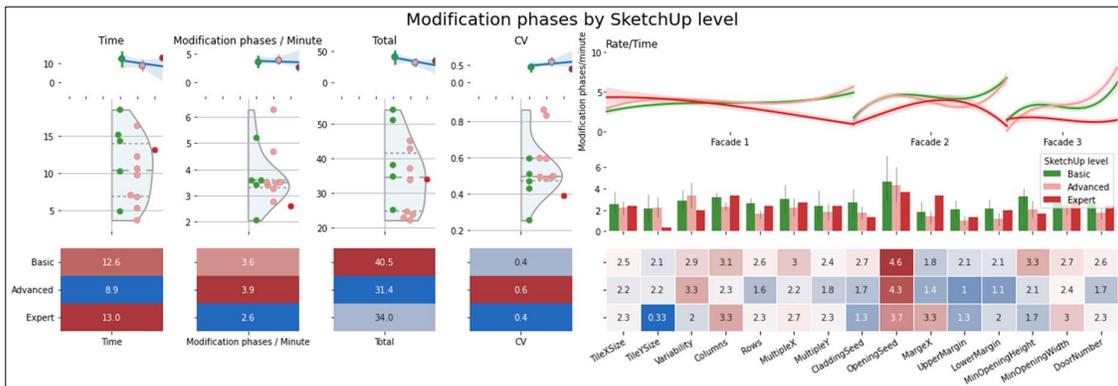
Appendix 2 : Average by SketchUp



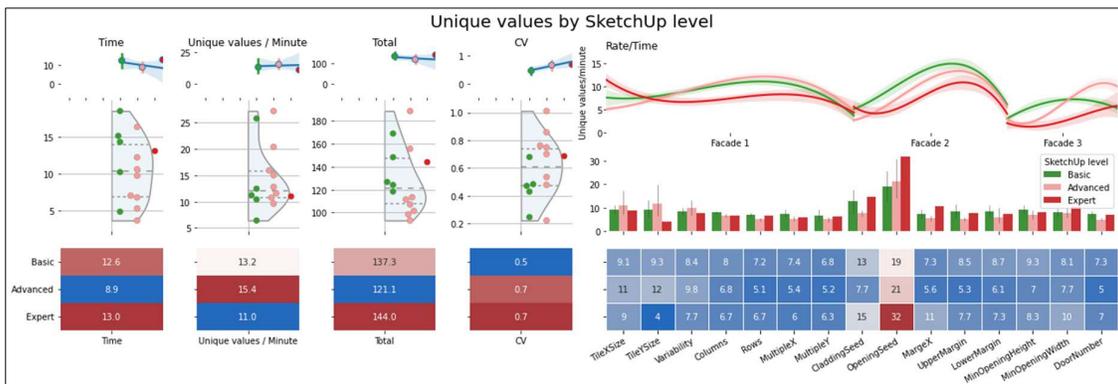
Appendix 2-1 : Average survey results by SketchUp



Appendix 2-2 : Iterations by SketchUp level

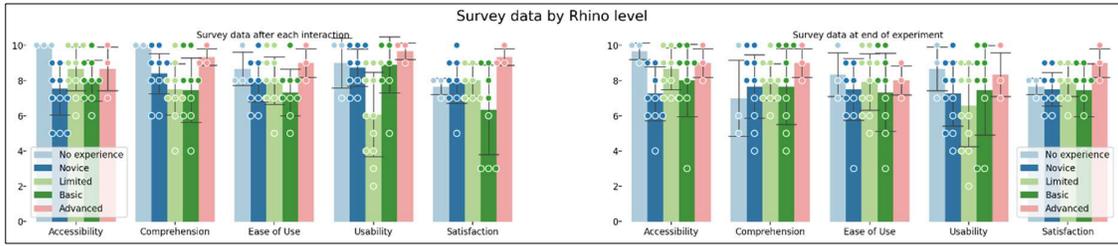


Appendix 2-3 : Modification phases by SketchUp level

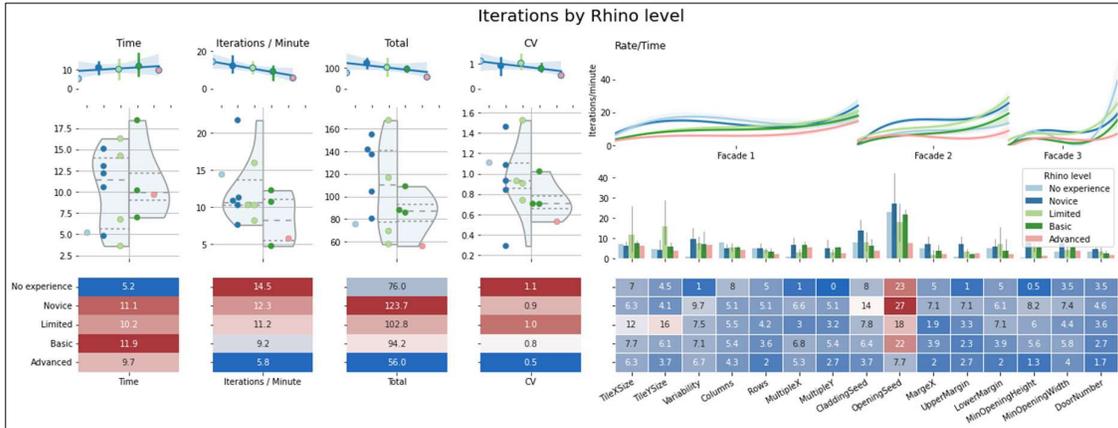


Appendix 2-4 : Unique values by SketchUp level

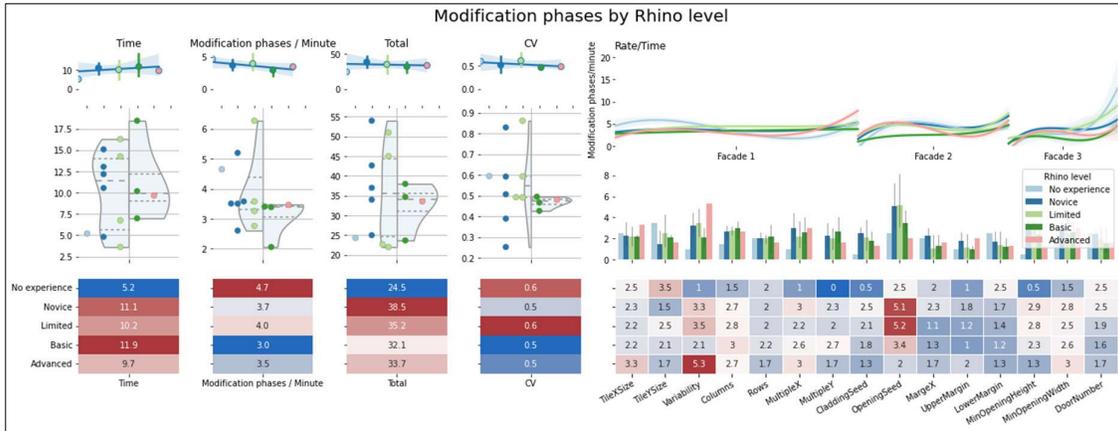
Appendix 3 : Average by Rhino



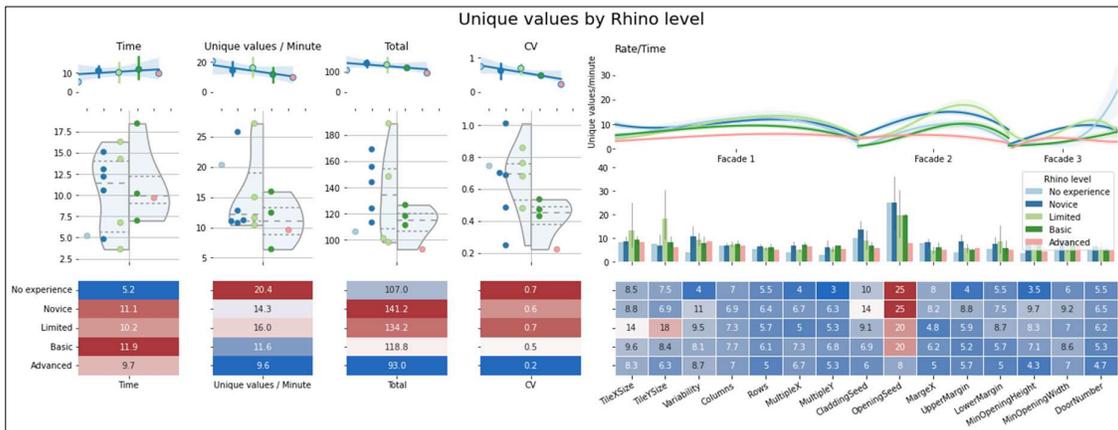
Appendix 3-1 : Survey data by Rhino level



Appendix 3-2 : Iterations by Rhino level

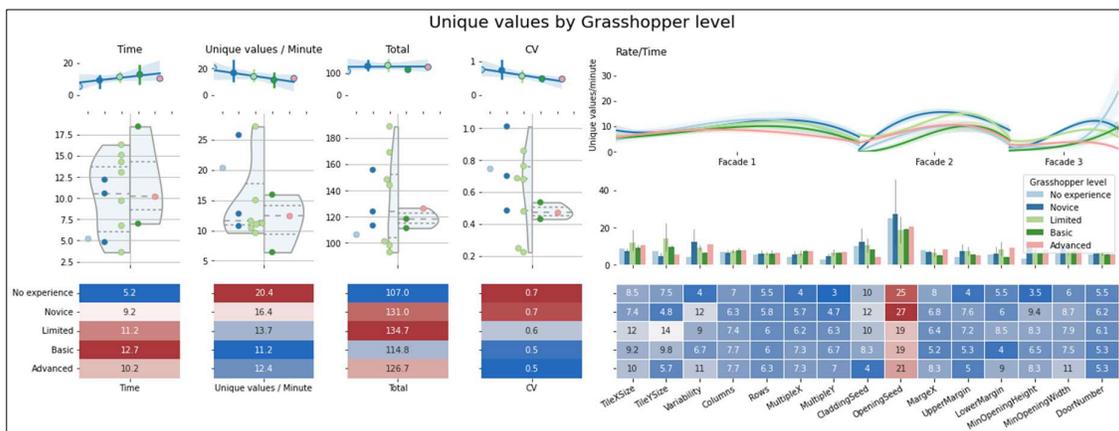
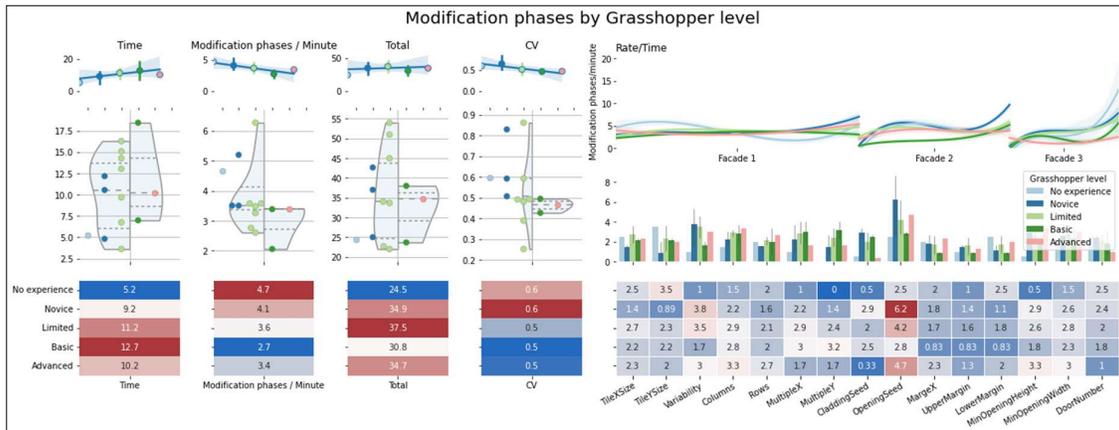
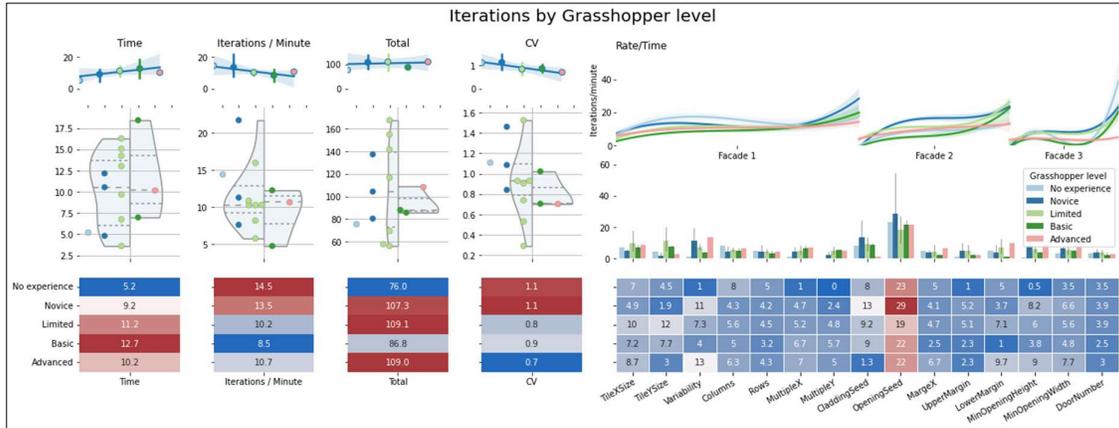
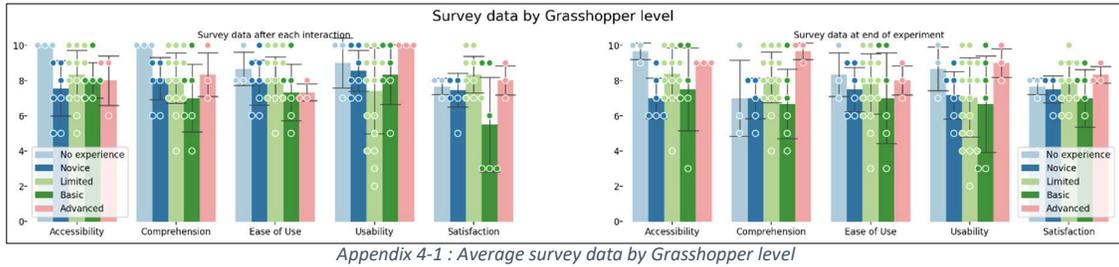


Appendix 3-3 : Modification phases by Rhino level

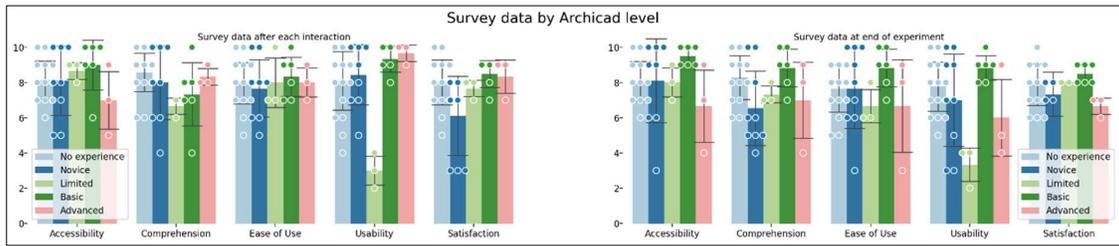


Appendix 3-4 : Unique values by Rhino level

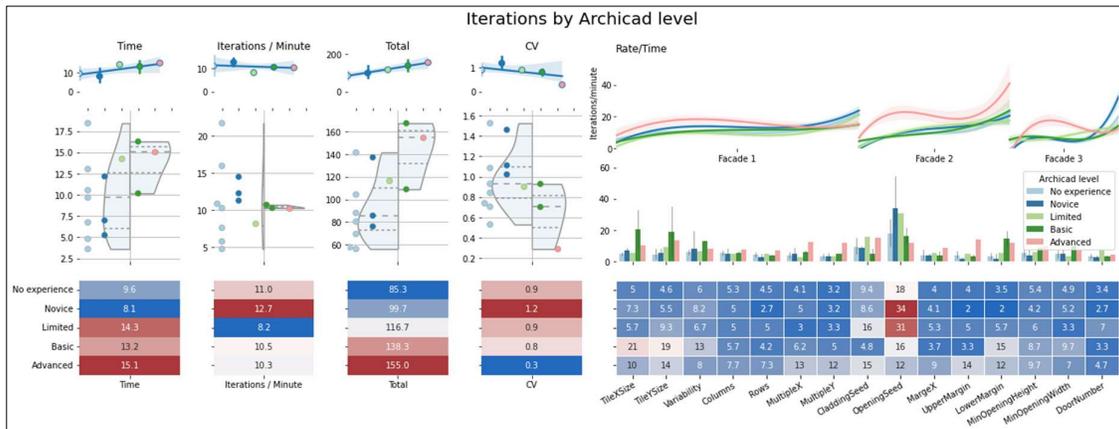
Appendix 4 : Average by Grasshopper



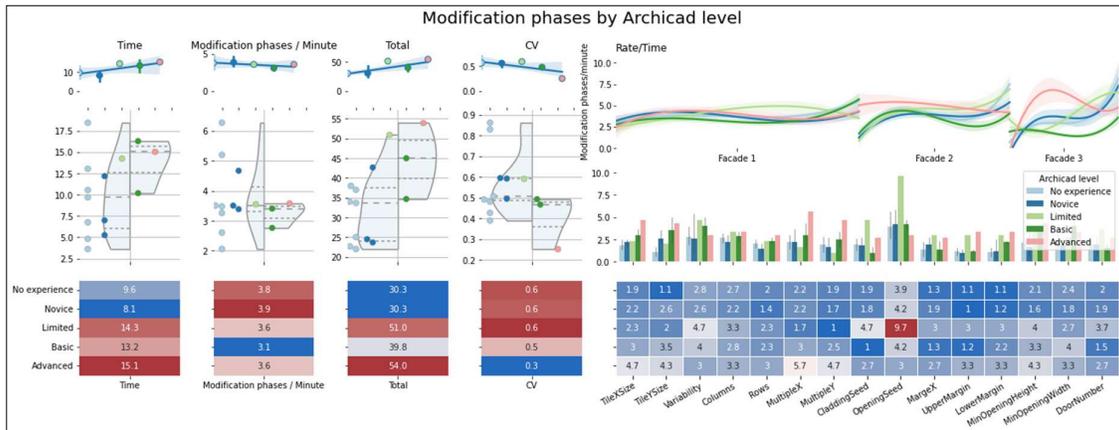
Appendix 5 : Average by Archicad



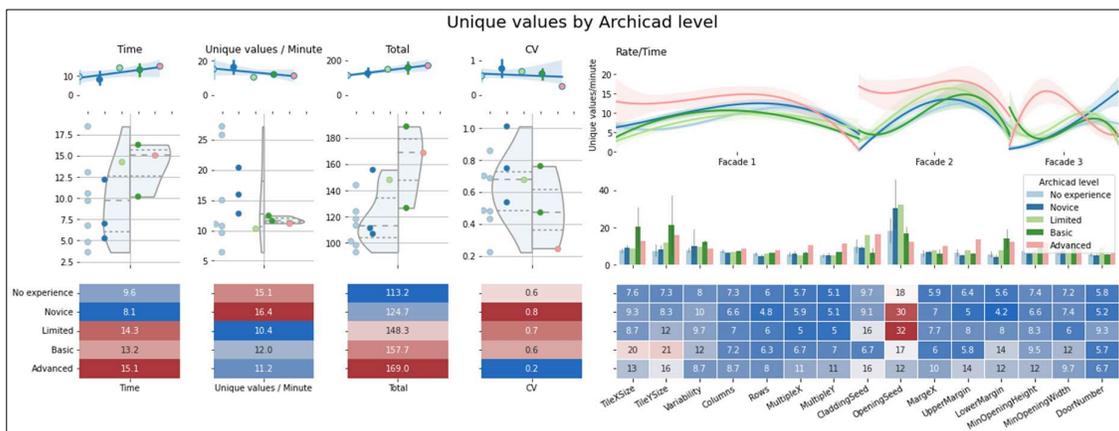
Appendix 5-1 : Average survey data by Archicad level



Appendix 5-2 : Iterations by Archicad level

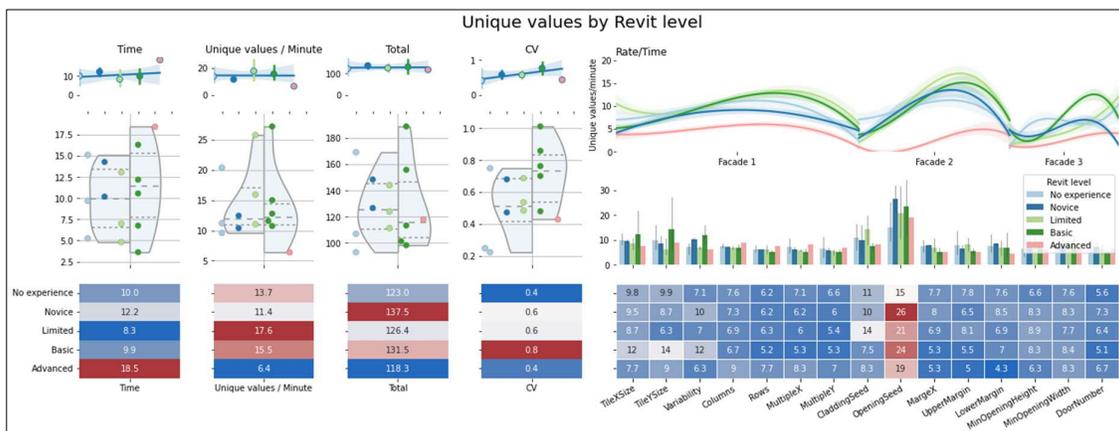
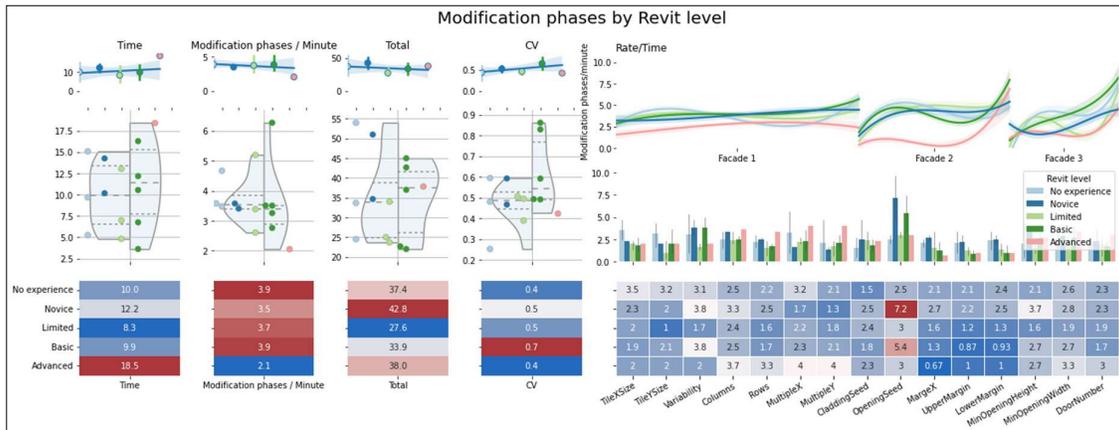
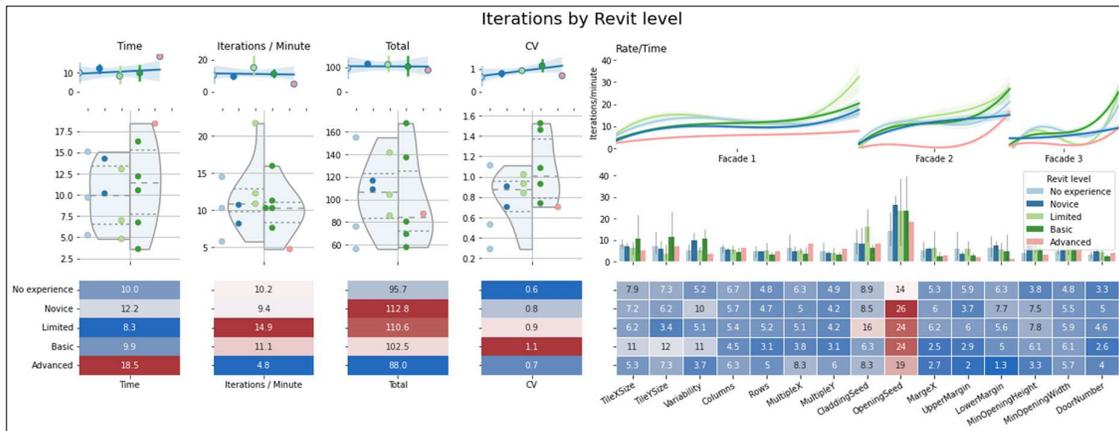
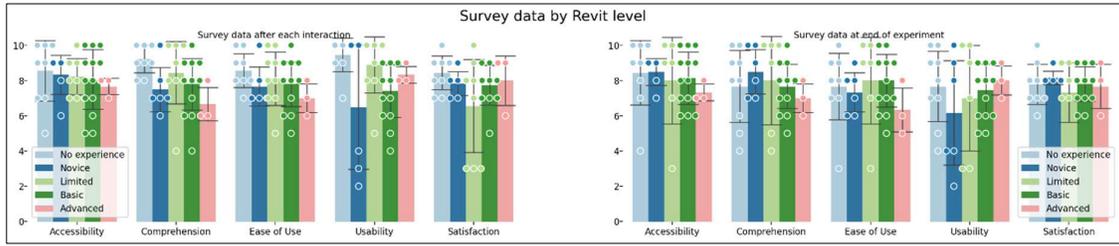


Appendix 5-3 : Modification phases by Archicad level

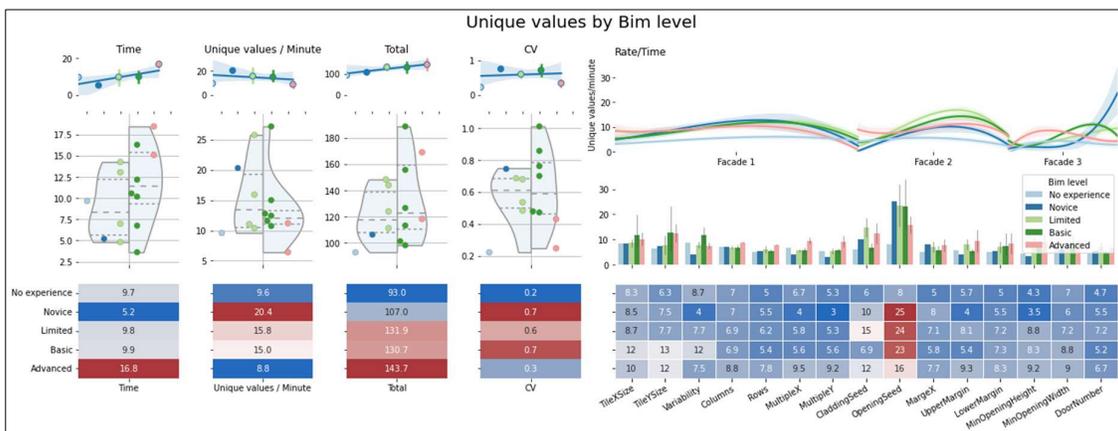
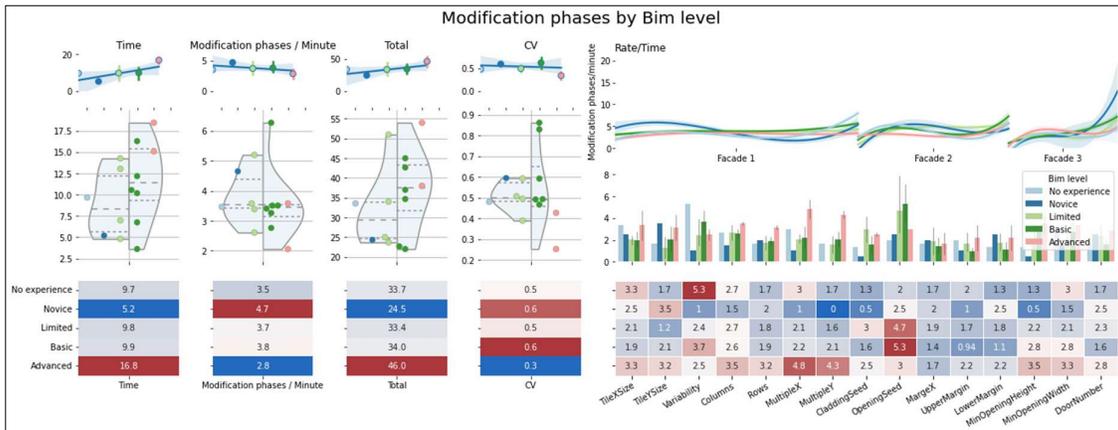
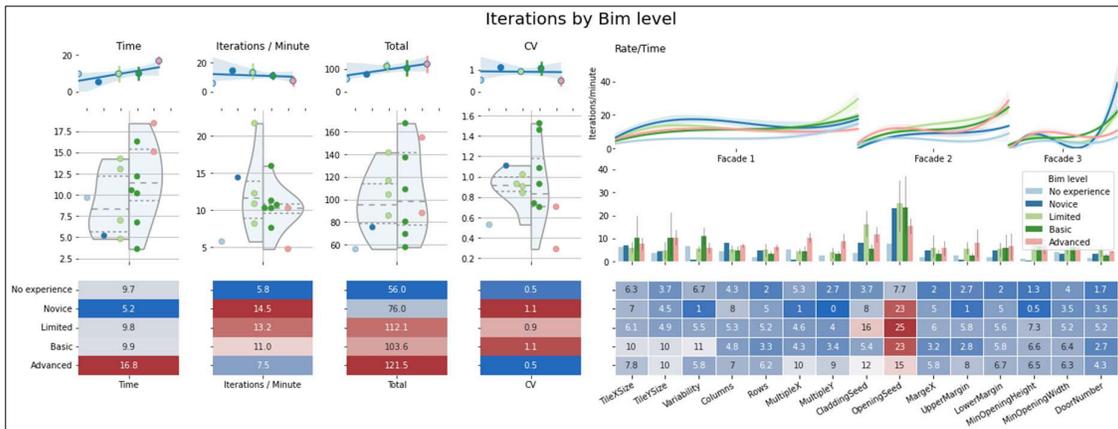
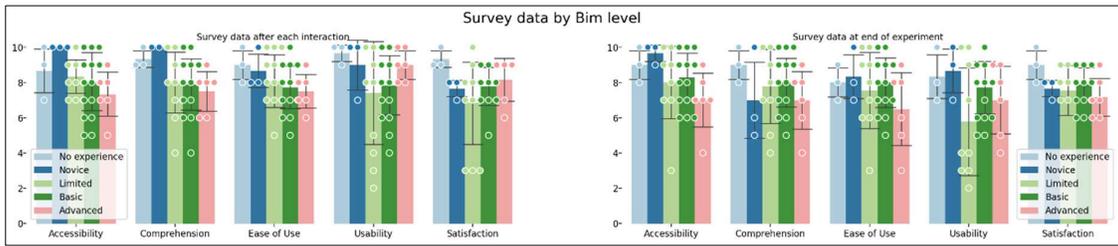


Appendix 5-4 : Unique values by Archicad level

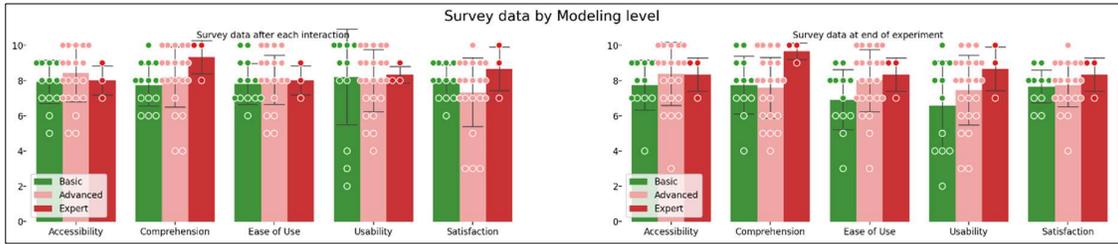
Appendix 6 : Average by Revit



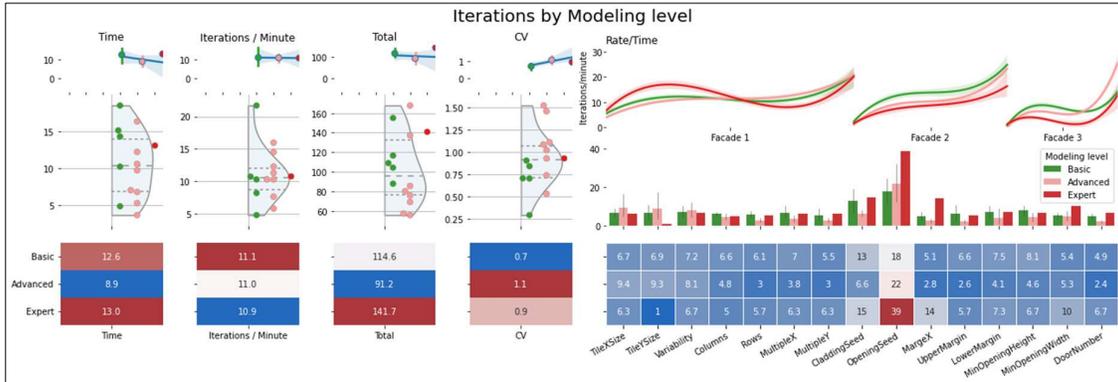
Appendix 7 : Average by BIM



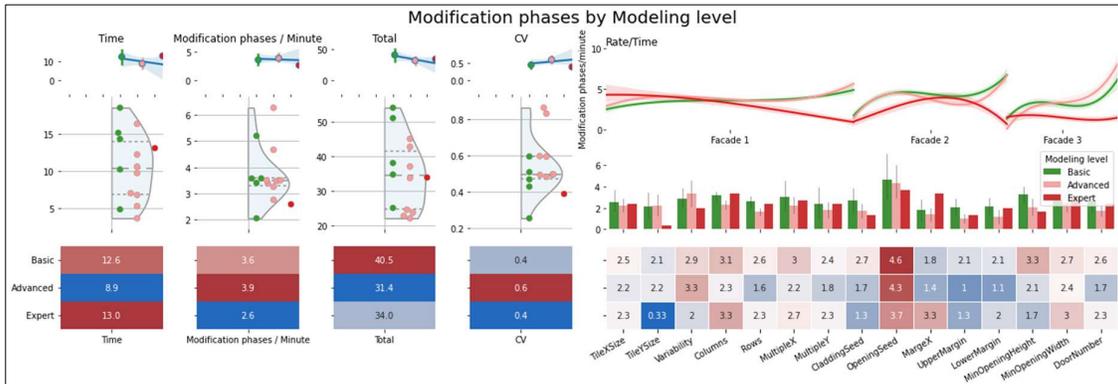
Appendix 8 : Average by Modeling



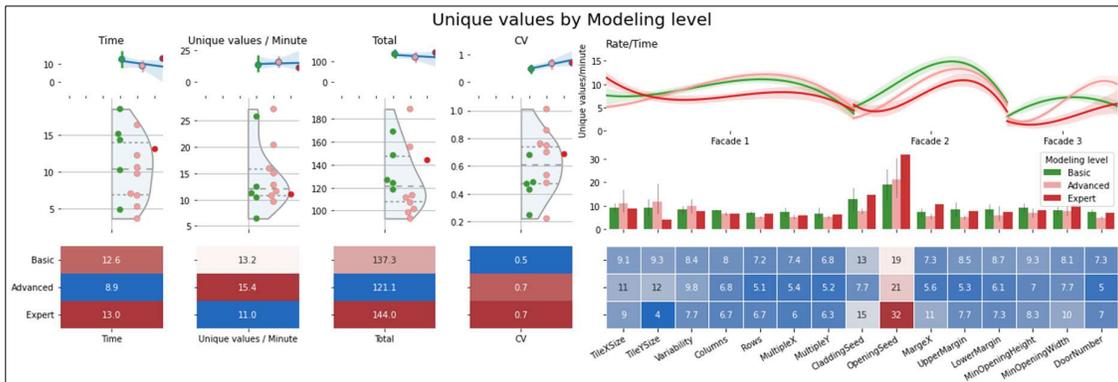
Appendix 8-1 : Survey data by [Modeling level](#)



Appendix 8-2 : Iterations by [Modeling level](#)

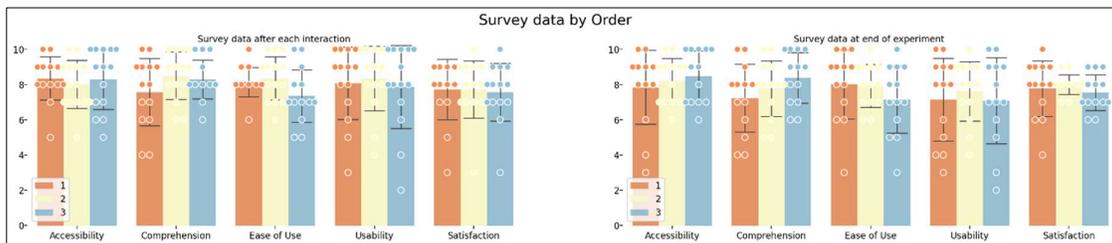


Appendix 8-3 : Modification phases by [Modeling level](#)

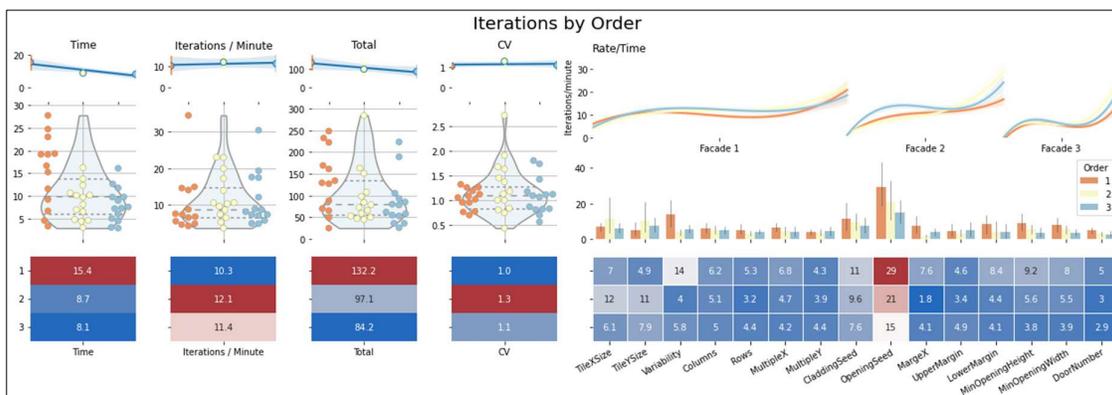


Appendix 8-4 : Unique values by [Modeling level](#)

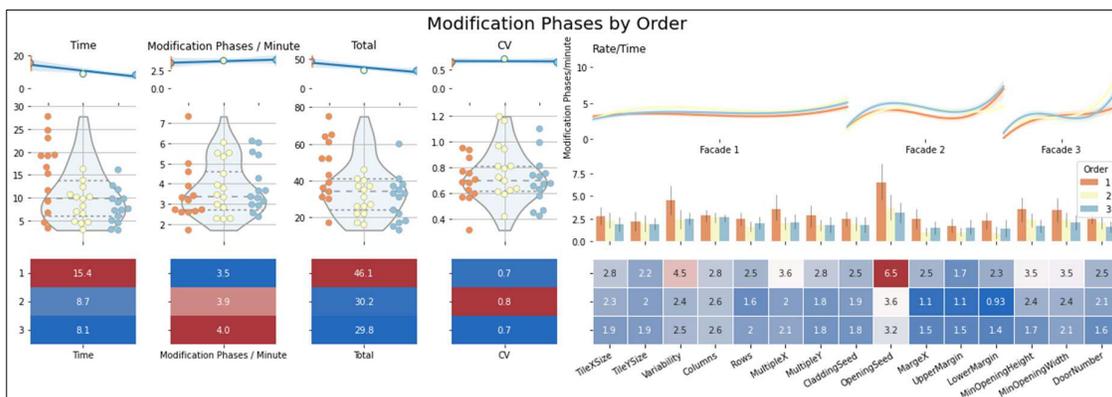
Appendix 9 : Average by Order



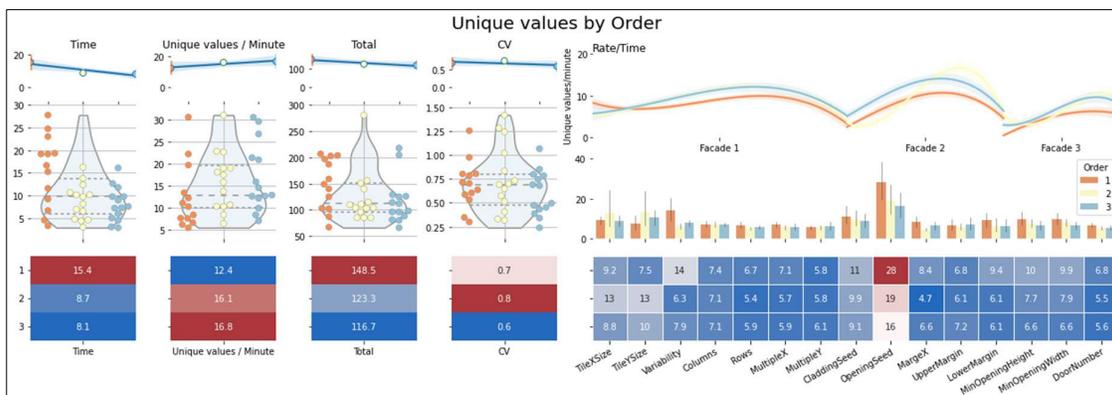
Appendix 9-1 : Survey results by order



Appendix 9-2 : Global Iterations by order

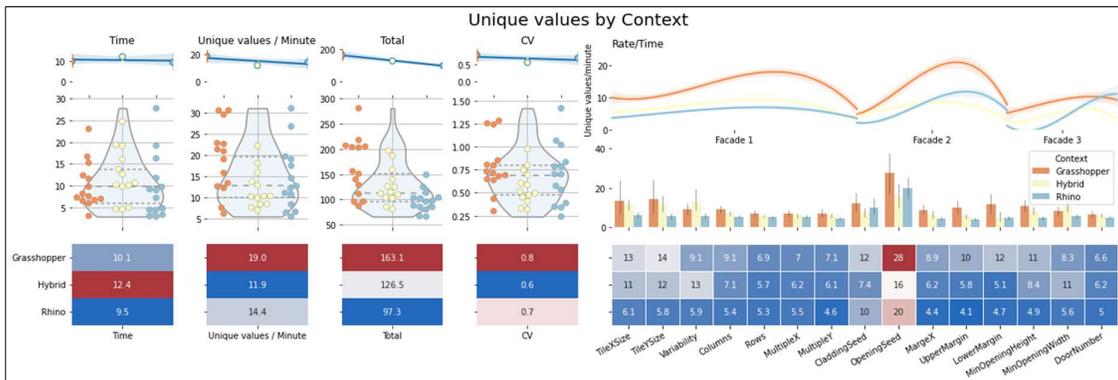
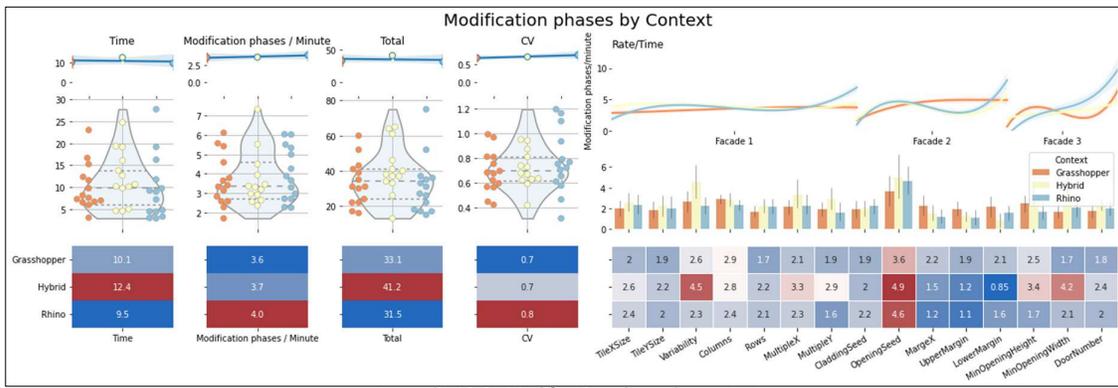
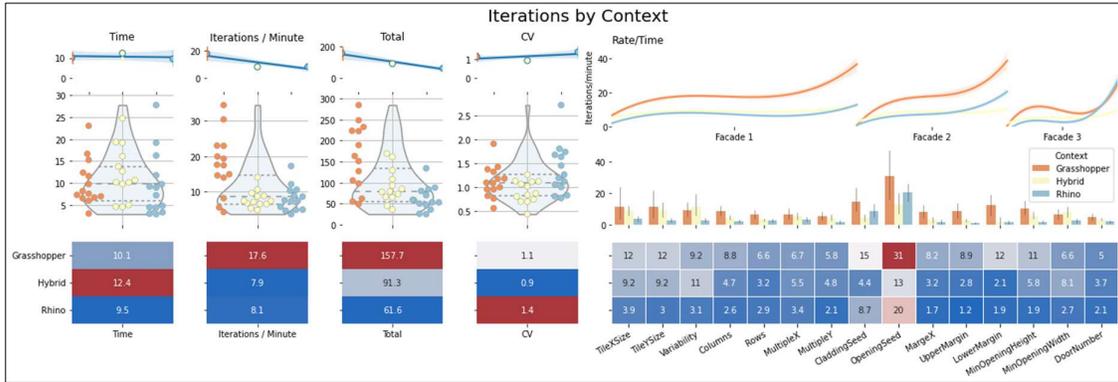
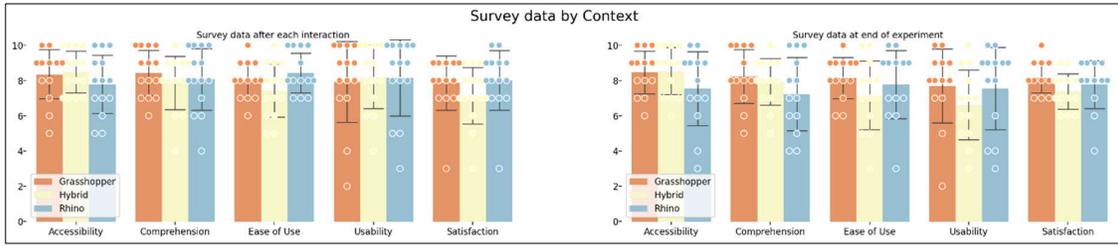


Appendix 9-3 : Modification phases by order

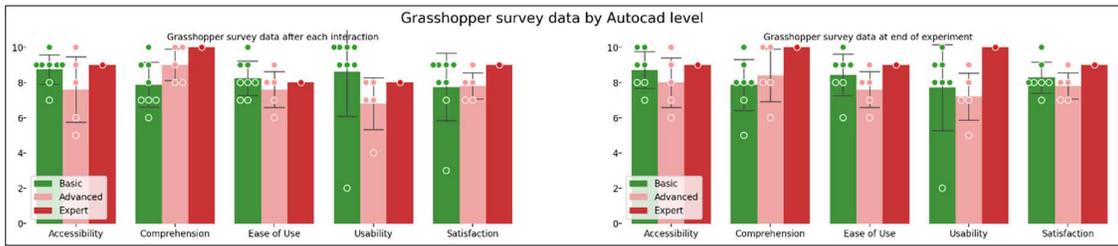


Appendix 9-4 : Unique values by order

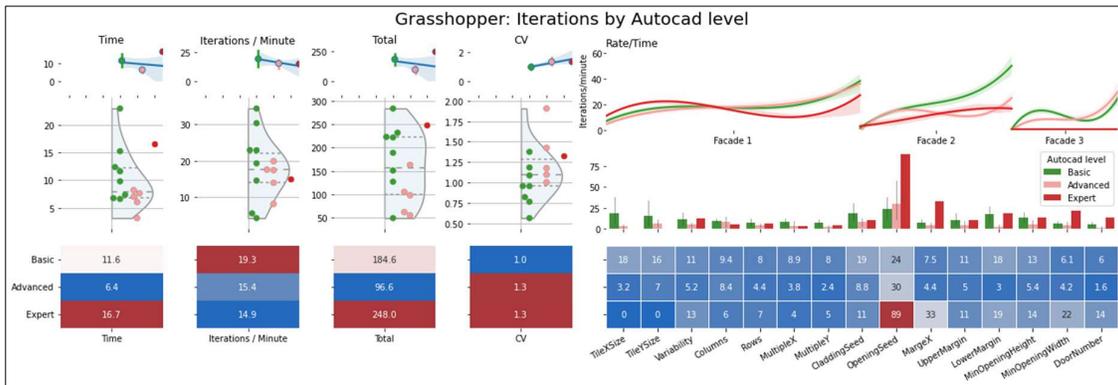
Appendix 10 : Average by Context



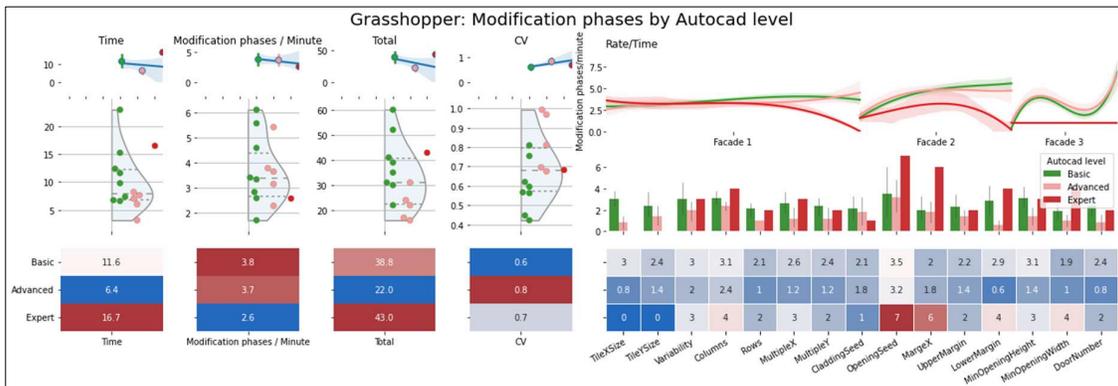
Appendix 11 : Grasshopper by AutoCAD



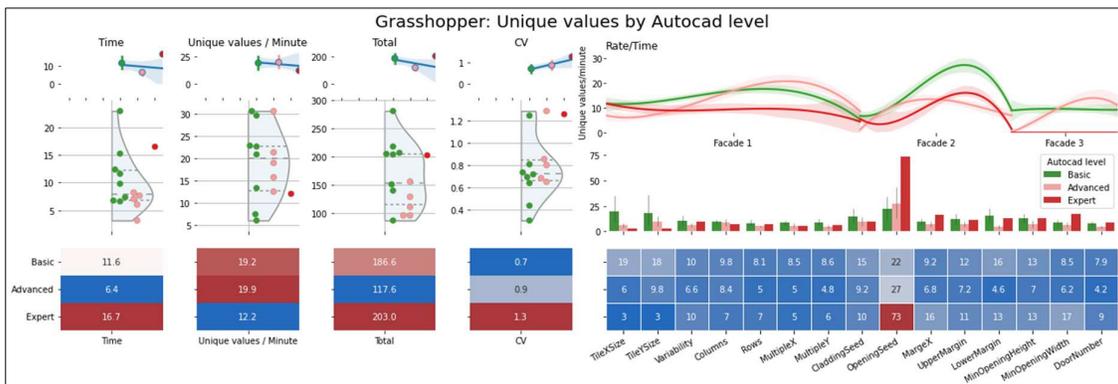
Appendix 11-1 : Grasshopper survey results by AutoCAD level



Appendix 11-2 : Grasshopper-Iterations by AutoCAD level

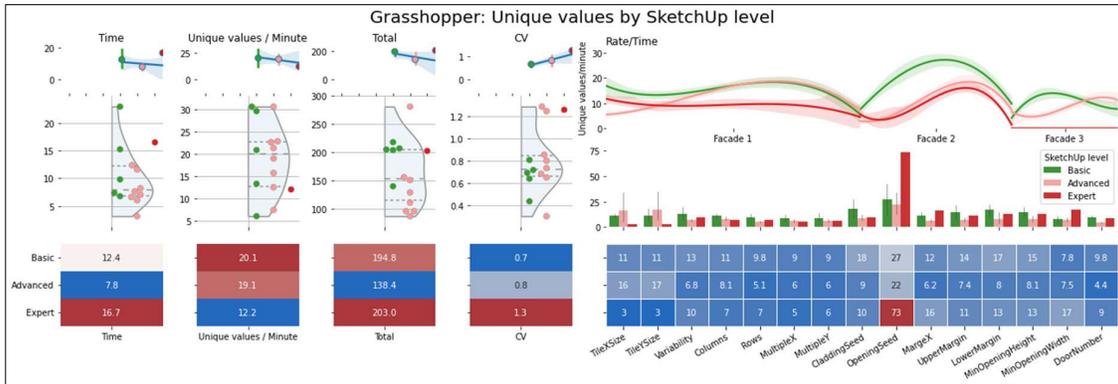
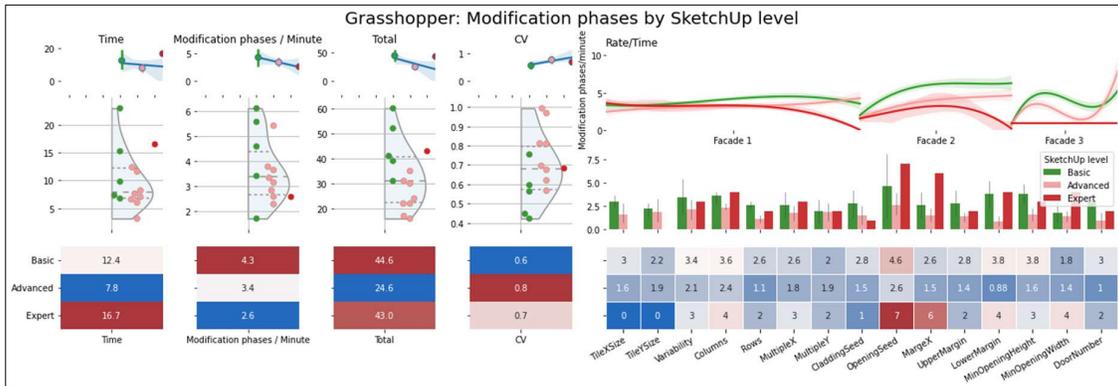
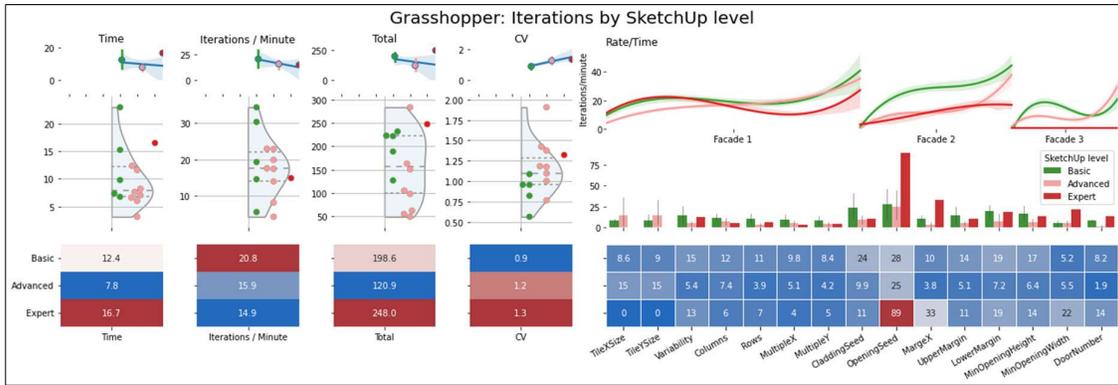
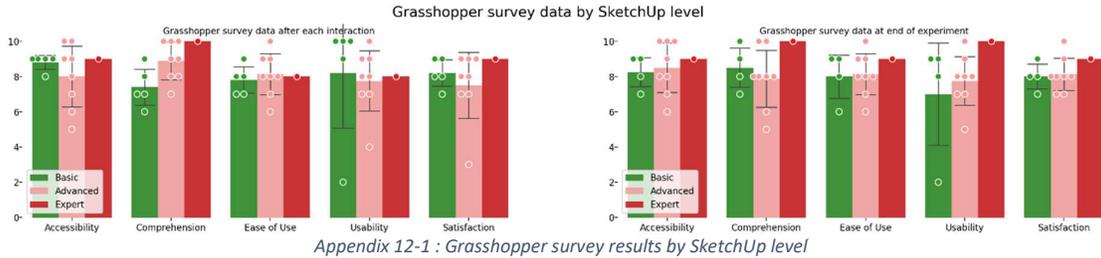


Appendix 11-3 : Grasshopper-Modification phases by AutoCAD level

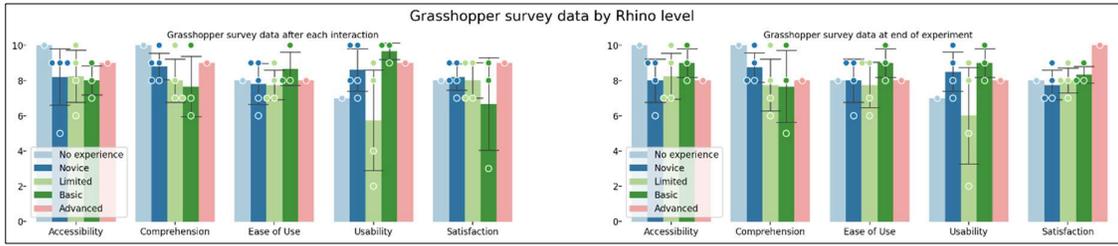


Appendix 11-4 : Grasshopper-Unique values by AutoCAD level

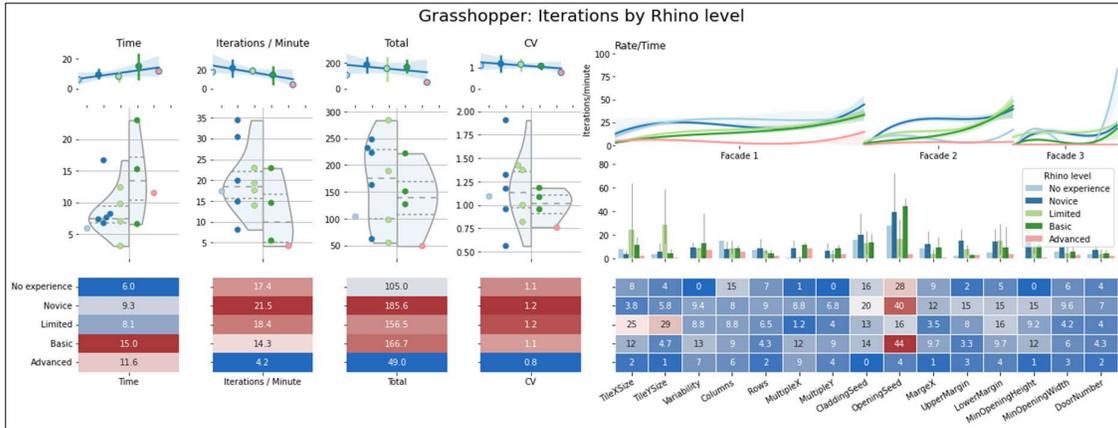
Appendix 12 : Grasshopper by SketchUp



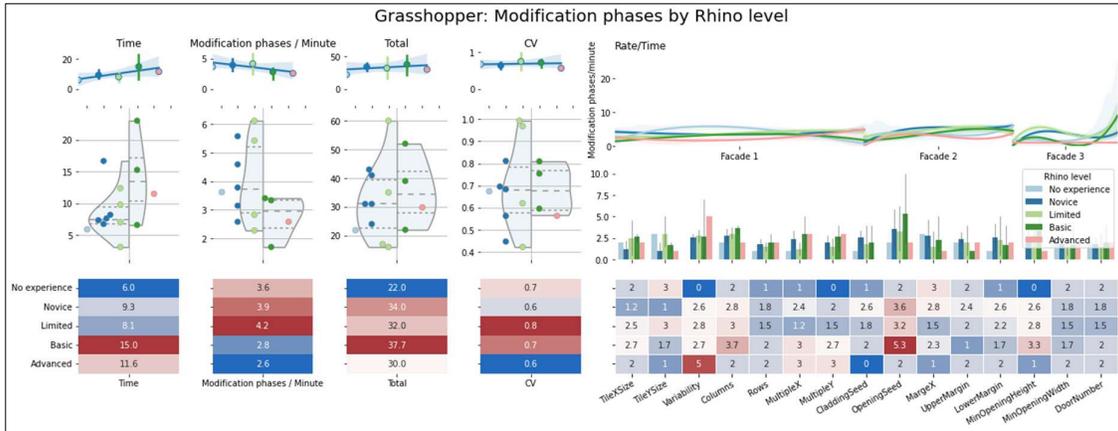
Appendix 13 : Grasshopper by Rhino



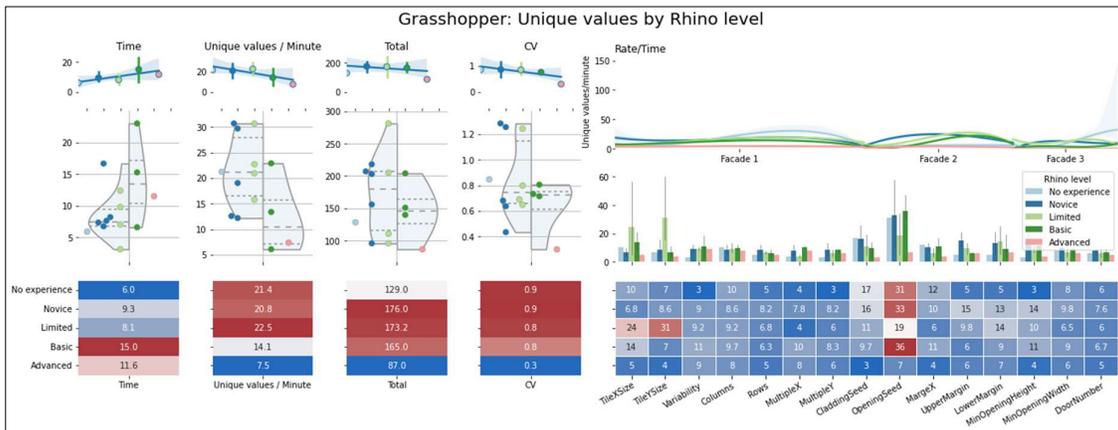
Appendix 13-1 : Grasshopper survey results by Rhino level



Appendix 13-2 : Grasshopper-Iterations by Rhino level

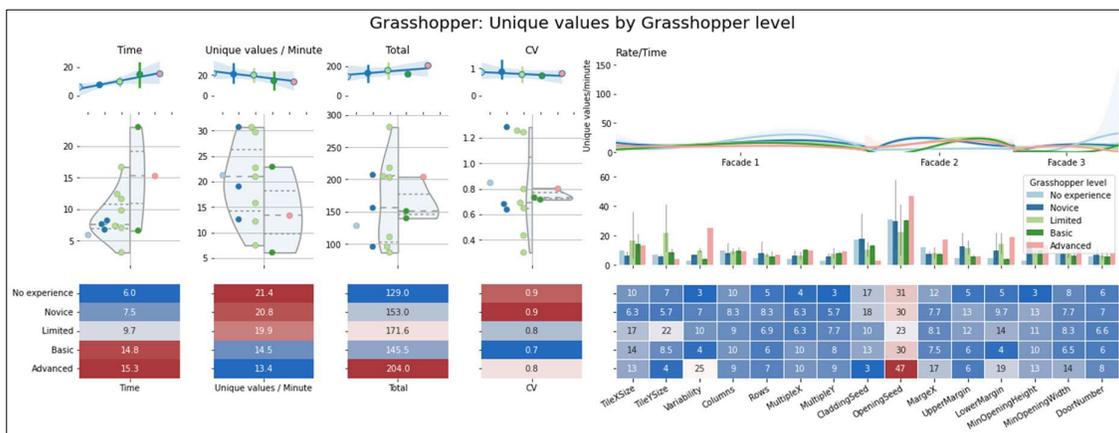
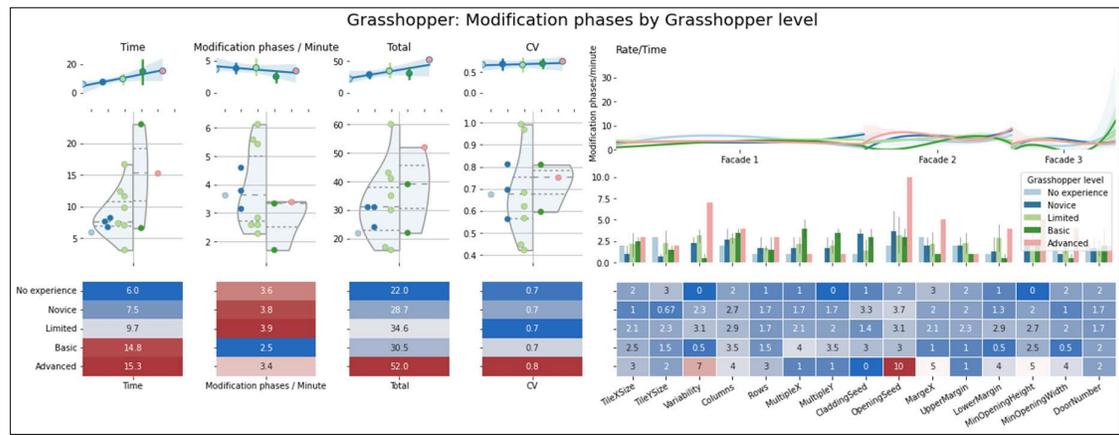
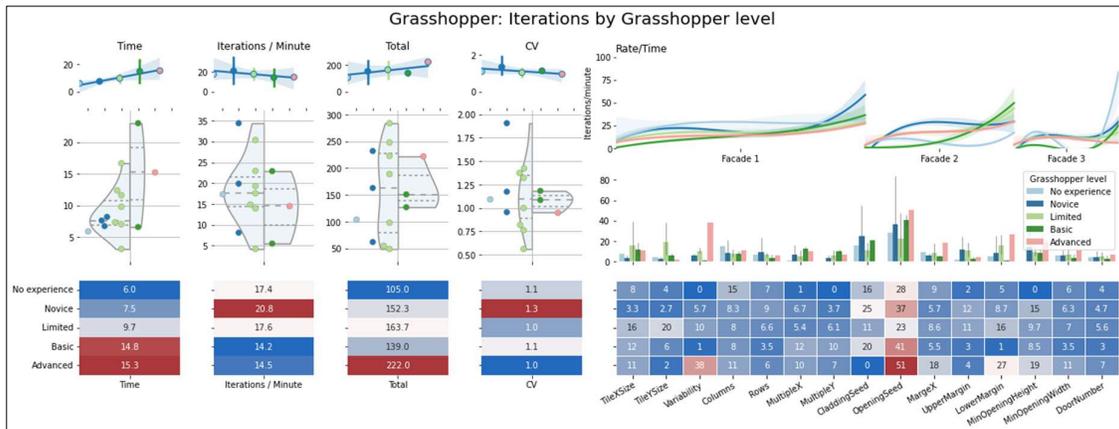
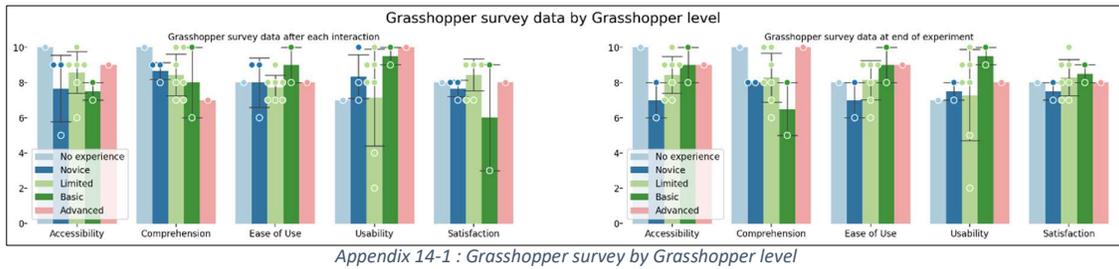


Appendix 13-3 : Grasshopper-Modification phases by Rhino level

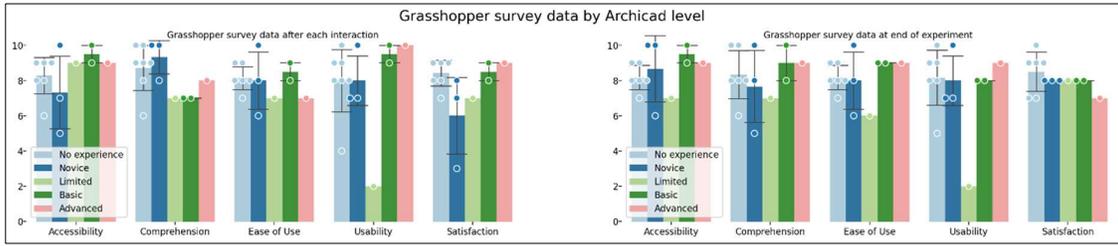


Appendix 13-4 : Grasshopper-Unique values by Rhino level

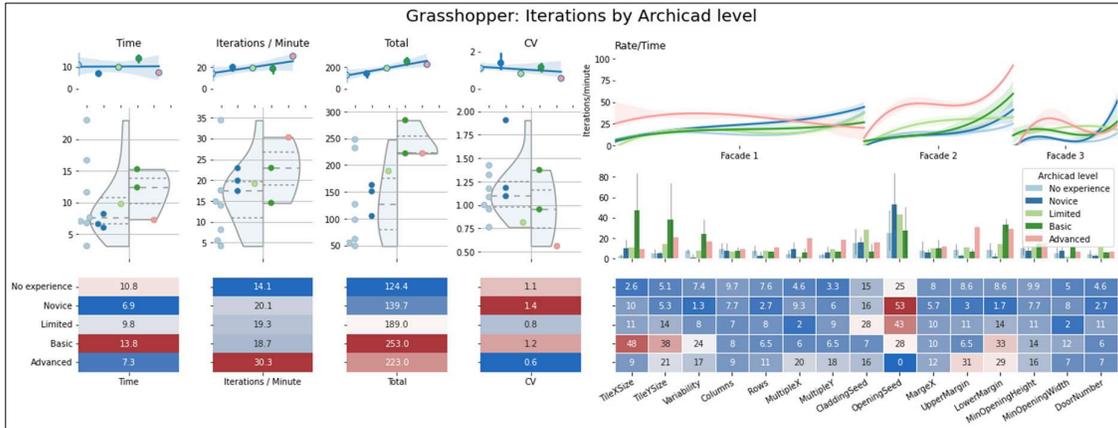
Appendix 14 : Grasshopper by Grasshopper



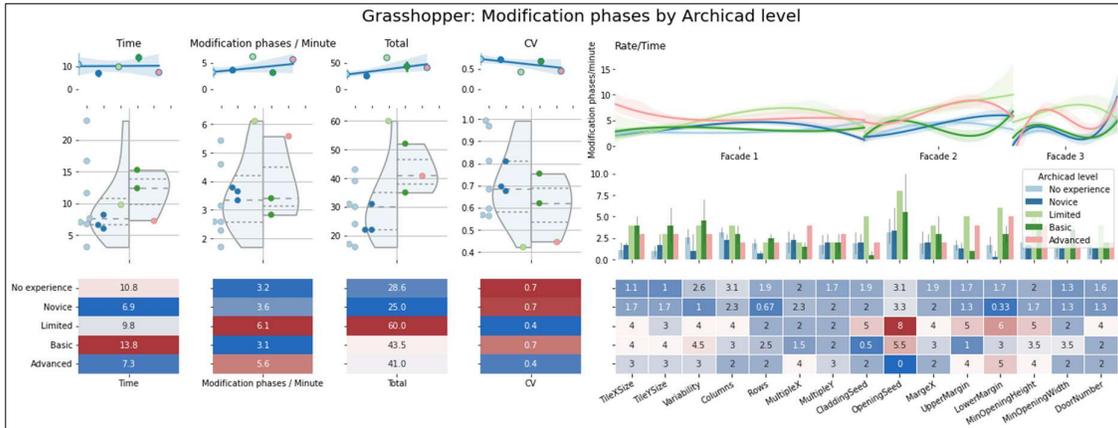
Appendix 15 : Grasshopper by Archicad



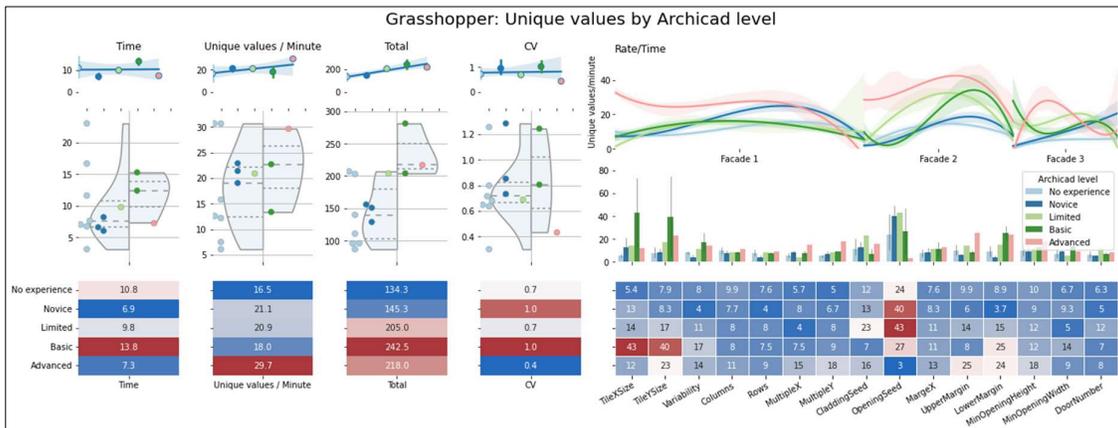
Appendix 15-1 : Grasshopper survey data by Archicad level



Appendix 15-2 : Grasshopper-Iterations by Archicad level

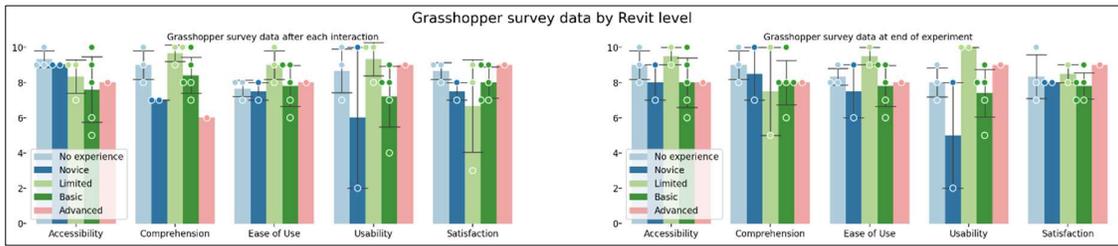


Appendix 15-3 : Grasshopper-Modification phases by Archicad level

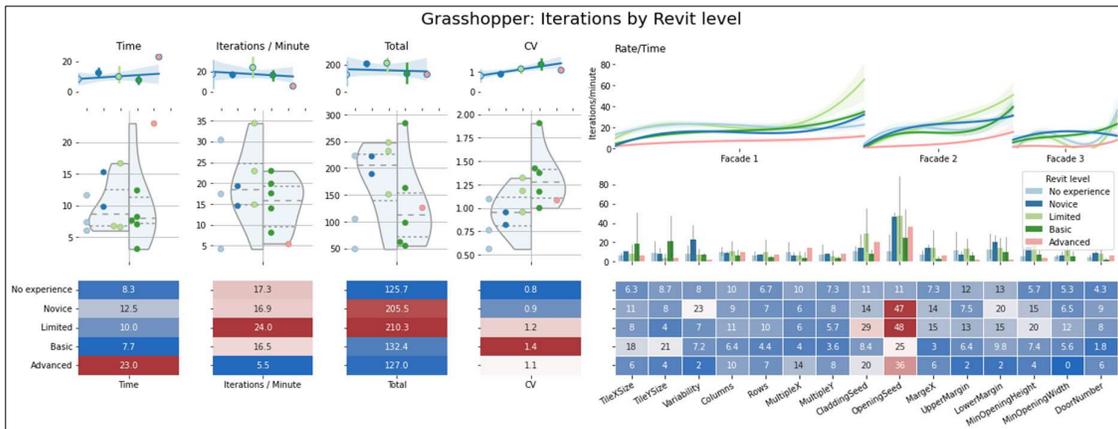


Appendix 15-4 : Grasshopper-Unique values by Archicad level

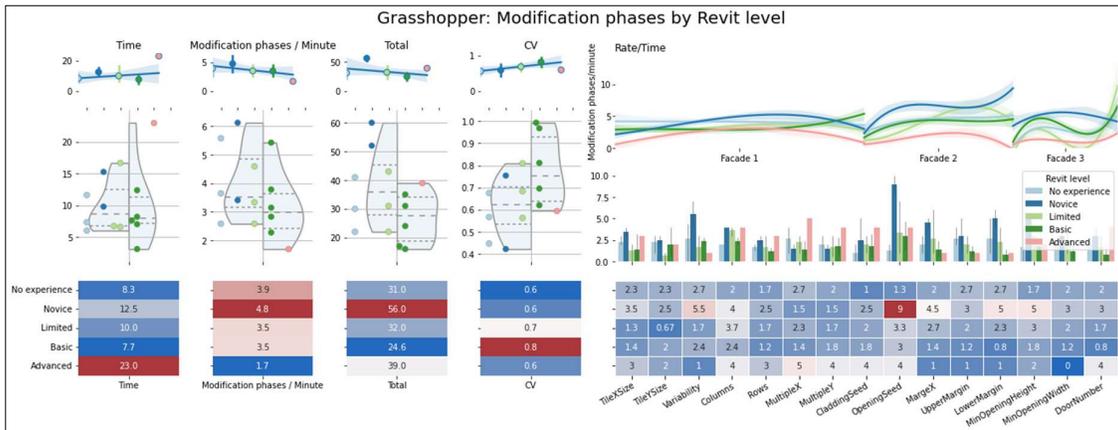
Appendix 16 : Grasshopper by Revit



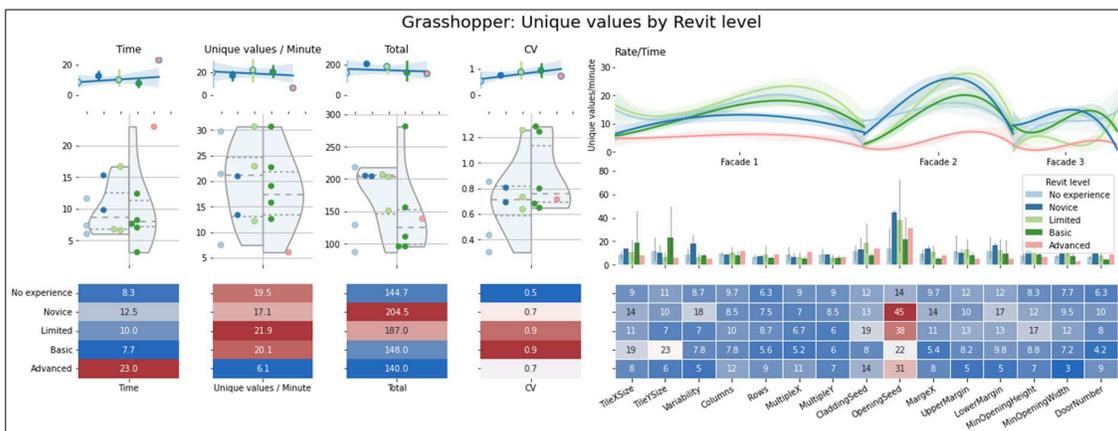
Appendix 16-1 : Grasshopper survey data by Revit level



Appendix 16-2 : Grasshopper-Iterations by Revit level

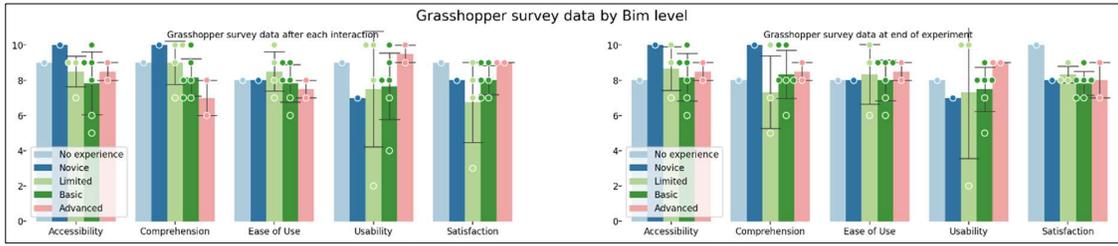


Appendix 16-3 : Grasshopper-Modification phases by Revit level

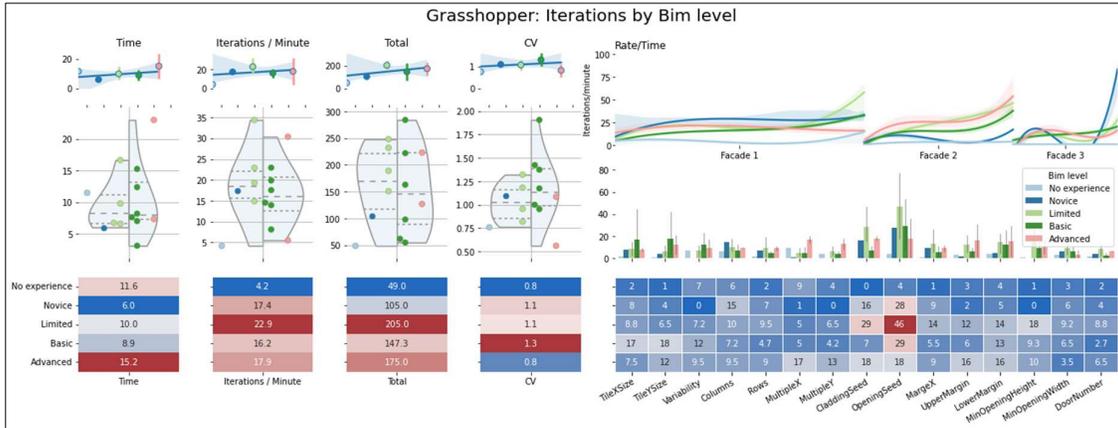


Appendix 16-4 : Grasshopper-Unique values by Revit level

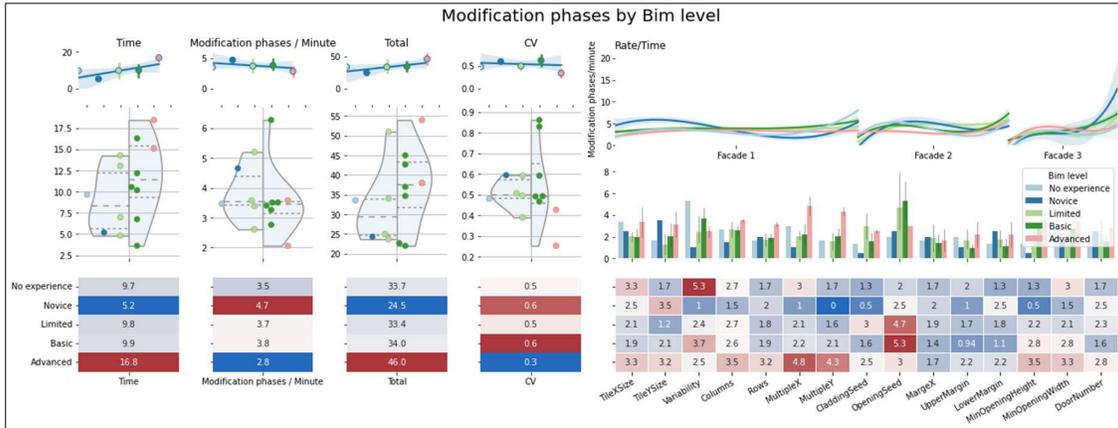
Appendix 17 : Grasshopper by BIM



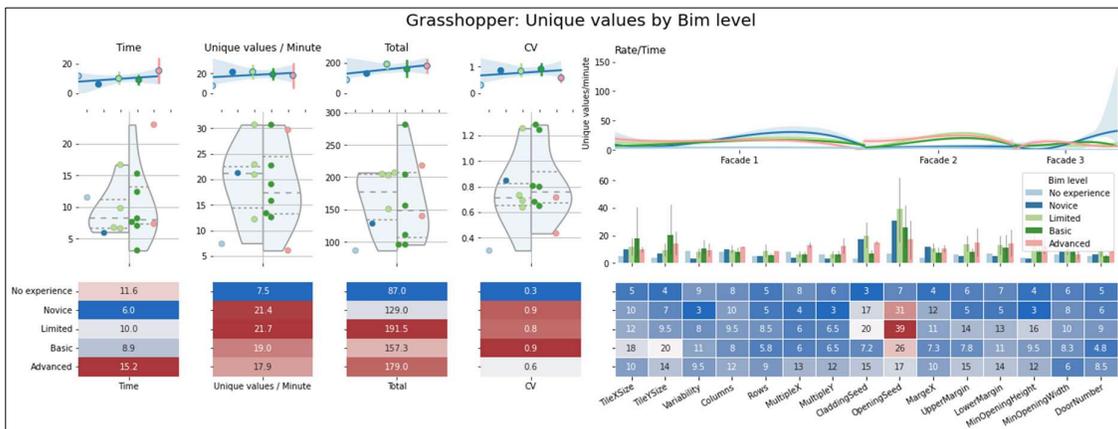
Appendix 17-1 : Grasshopper survey data by BIM level



Appendix 17-2 : Grasshopper-Iterations by BIM level

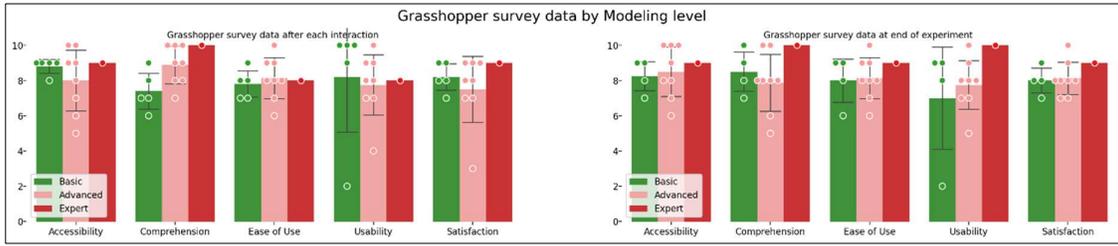


Appendix 17-3 : Grasshopper-Modification phases by BIM level

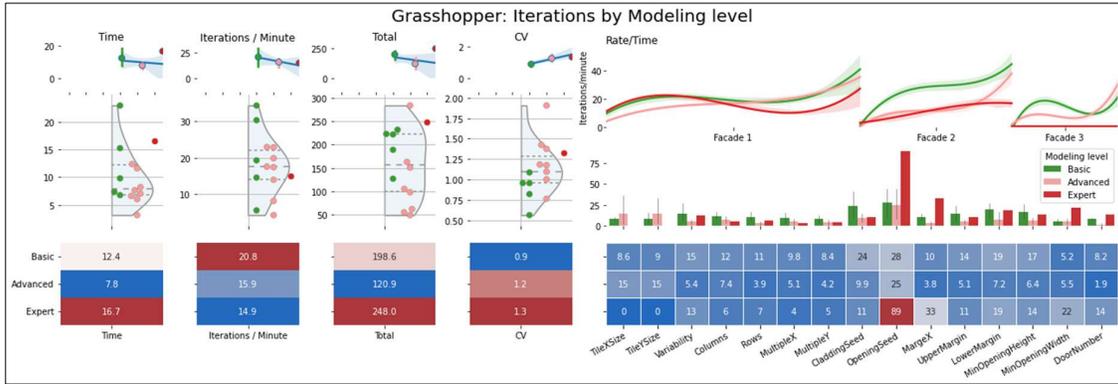


Appendix 17-4 : Grasshopper-Unique values by BIM level

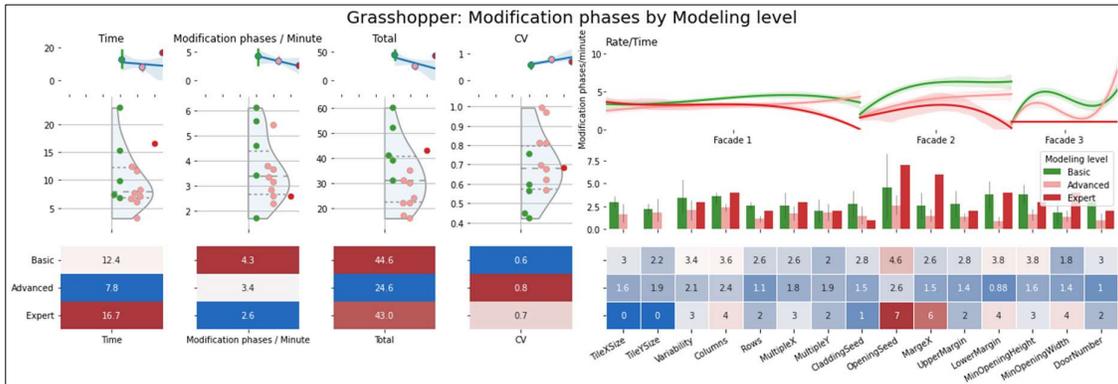
Appendix 18 : Grasshopper by Modeling



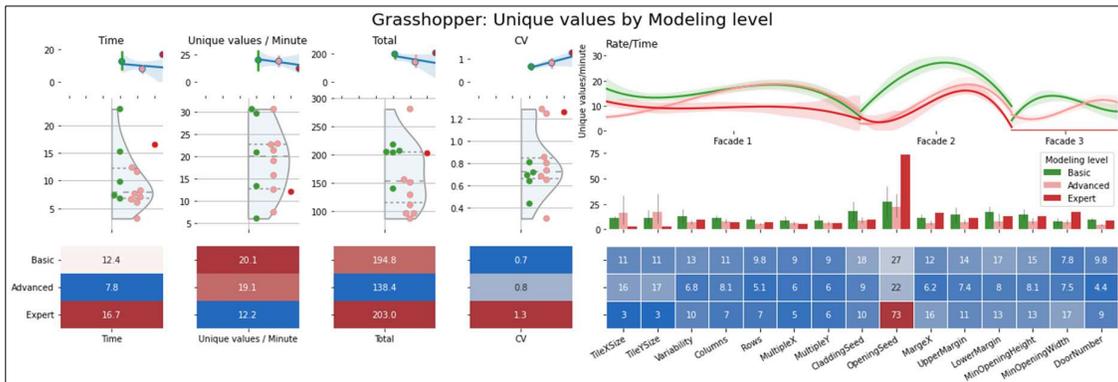
Appendix 18-1 : Grasshopper survey data by [Modeling level](#)



Appendix 18-2 : Grasshopper-Iterations by [Modeling level](#)

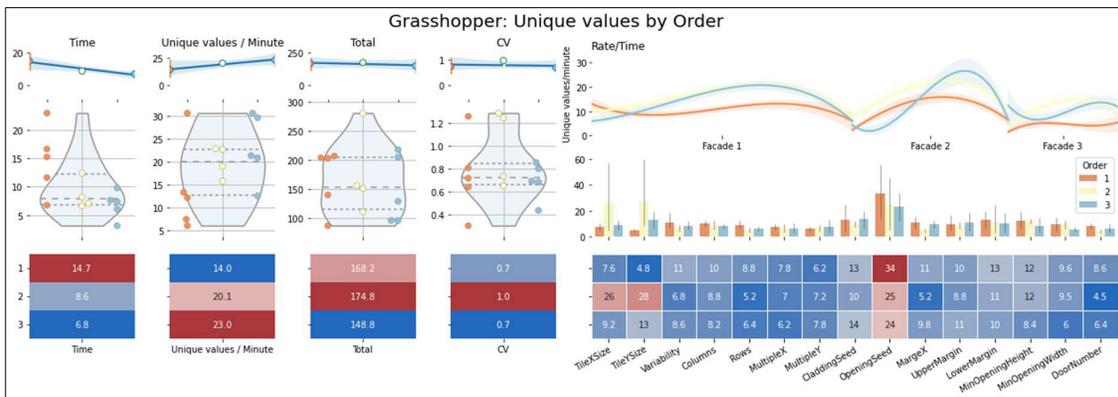
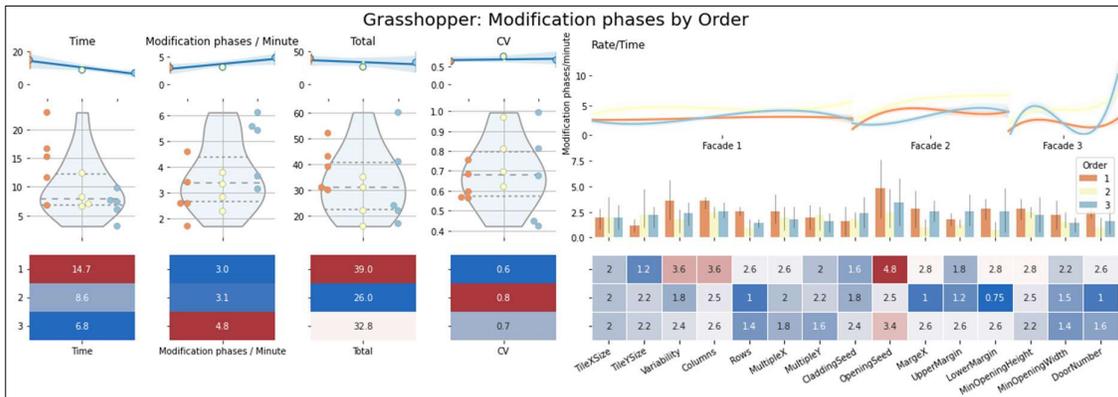
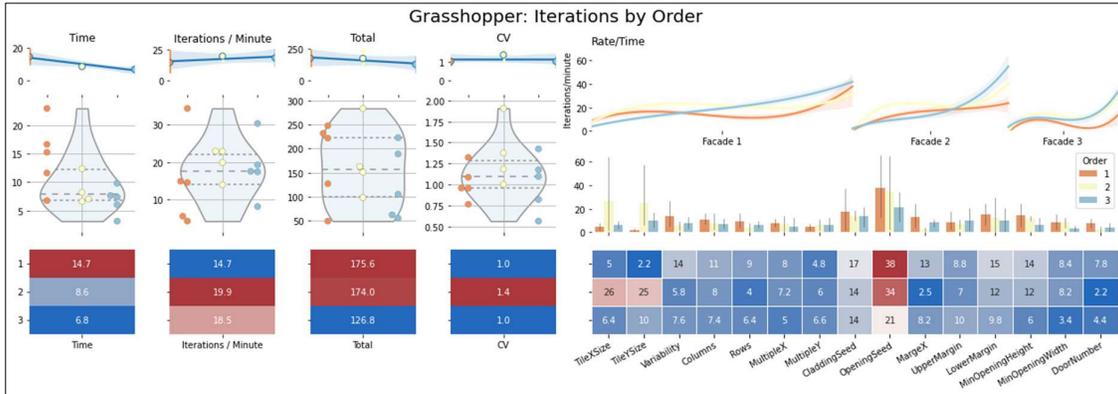
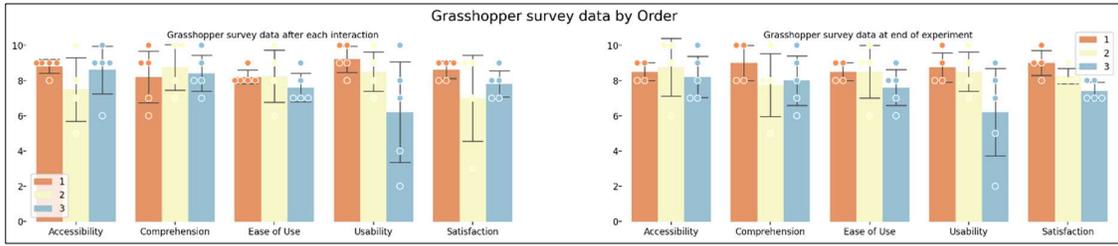


Appendix 18-3 : Grasshopper-Modification phases by [Modeling level](#)

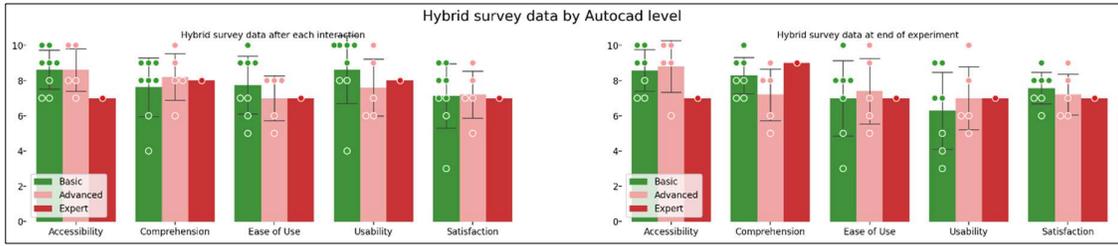


Appendix 18-4 : Grasshopper-Unique values by [Modeling level](#)

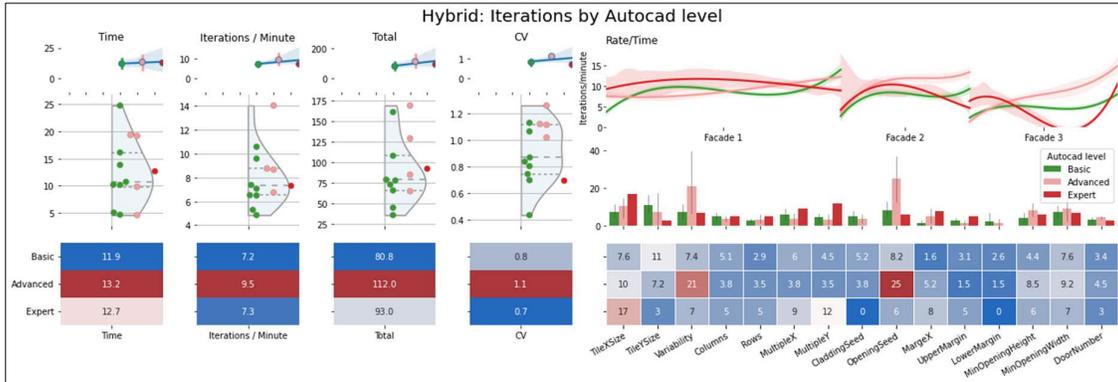
Appendix 19 : Grasshopper by Order



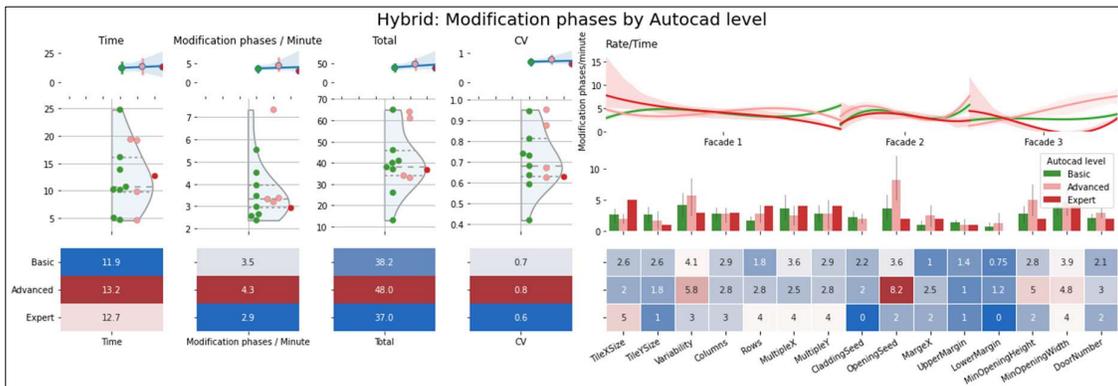
Appendix 20 : Hybrid by AutoCAD



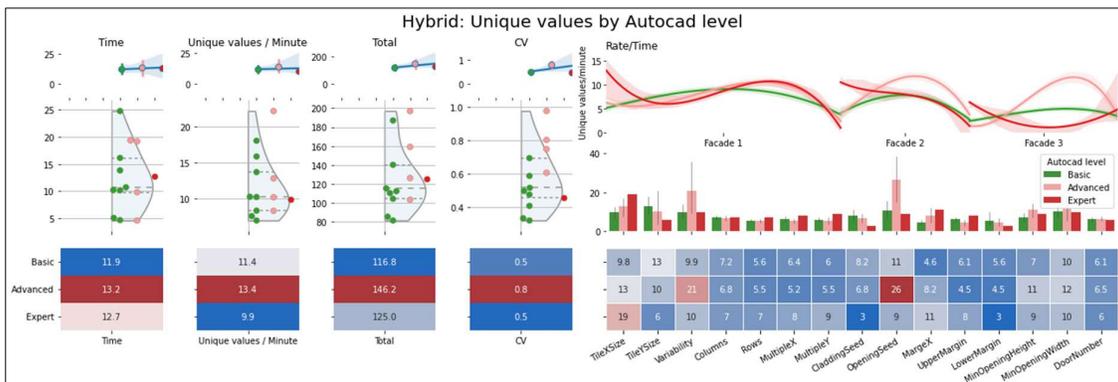
Appendix 20-1 : Hybrid survey data by [AutoCAD level](#)



Appendix 20-2 : Hybrid-Iterations by [AutoCAD level](#)

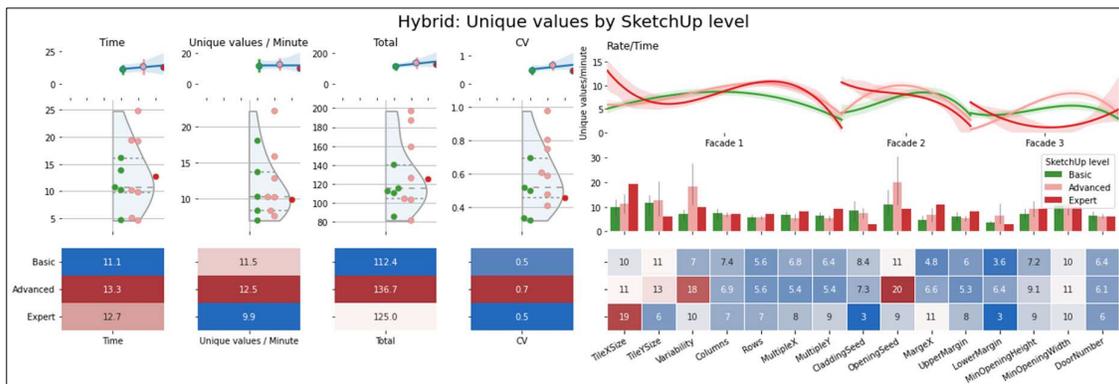
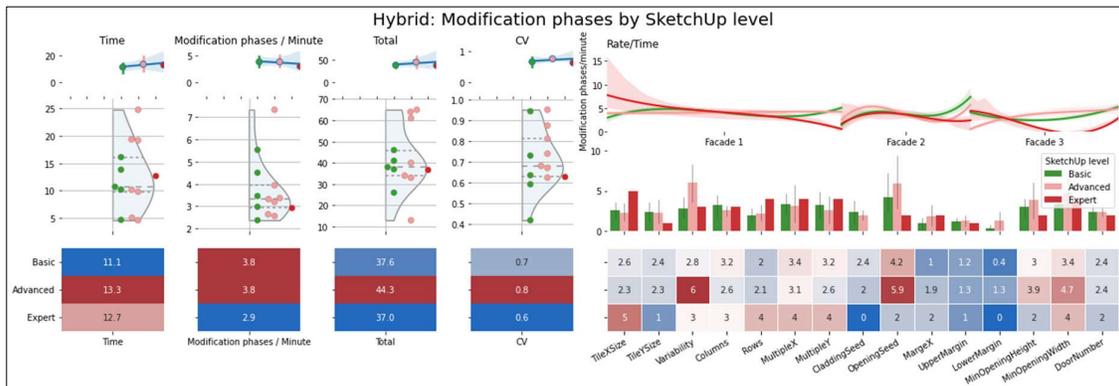
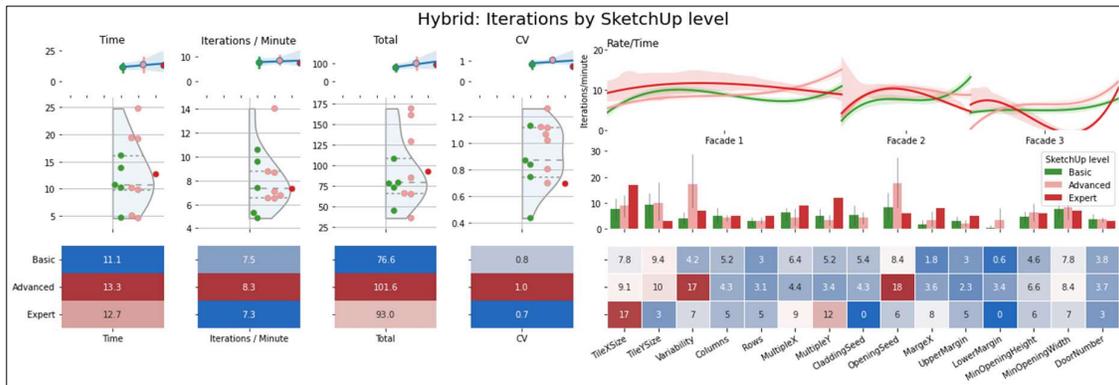
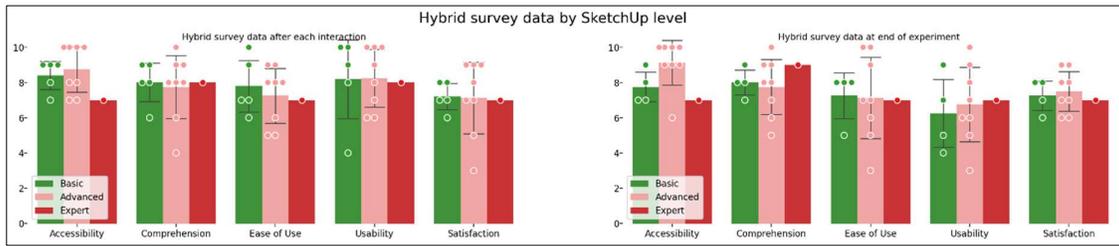


Appendix 20-3 : Hybrid-Modification phases by [AutoCAD level](#)

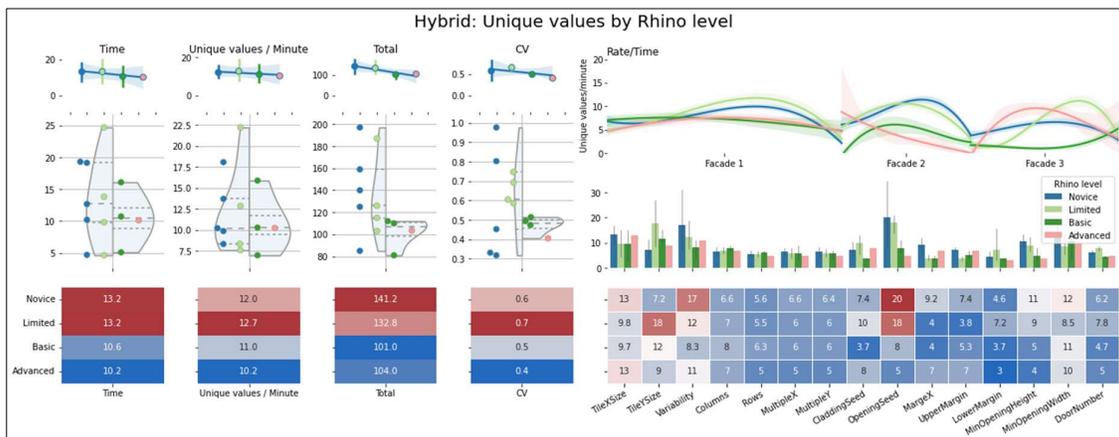
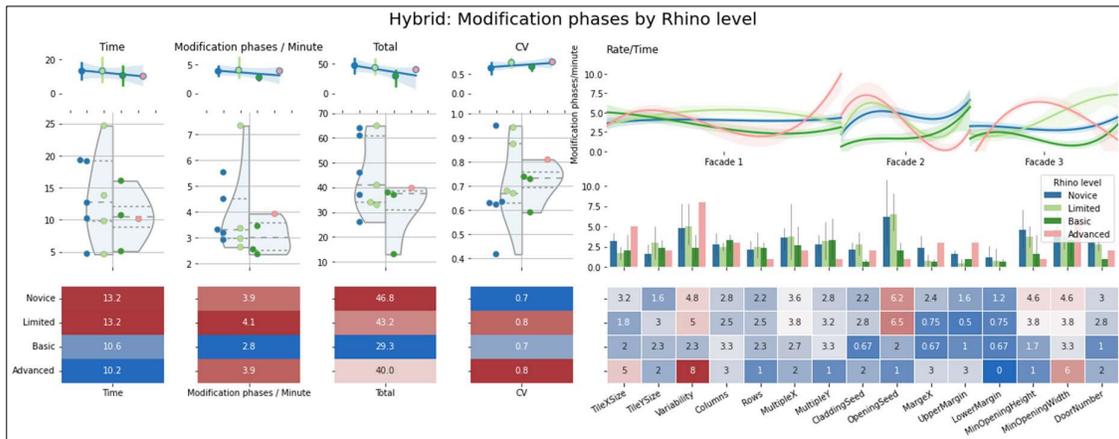
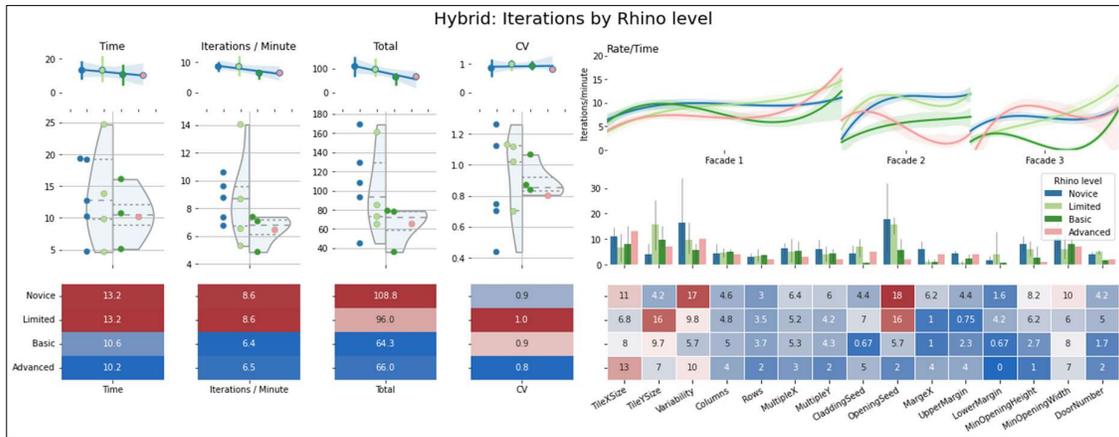
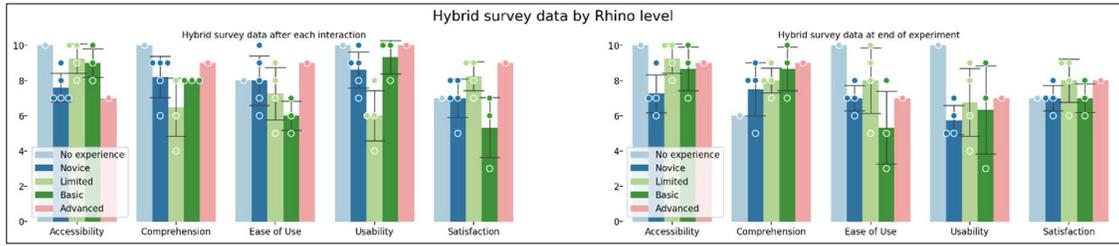


Appendix 20-4 : Hybrid-Unique values by [AutoCAD level](#)

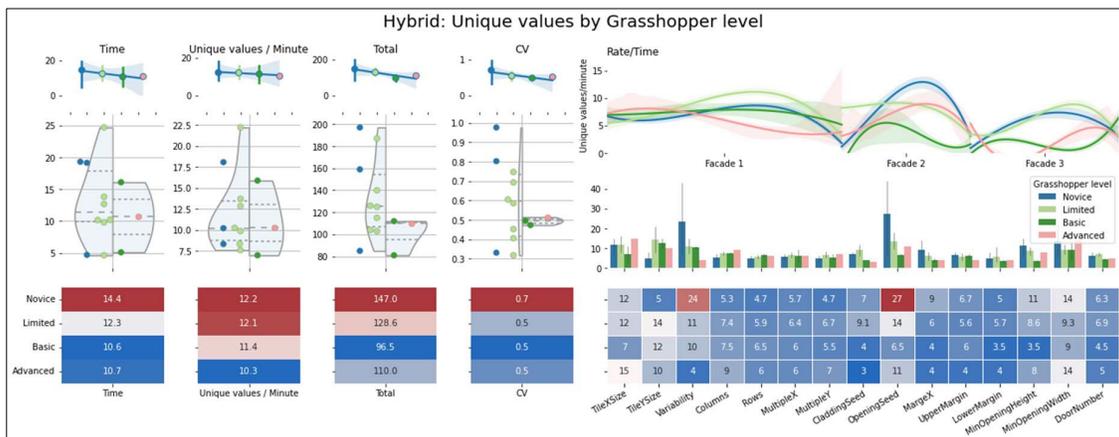
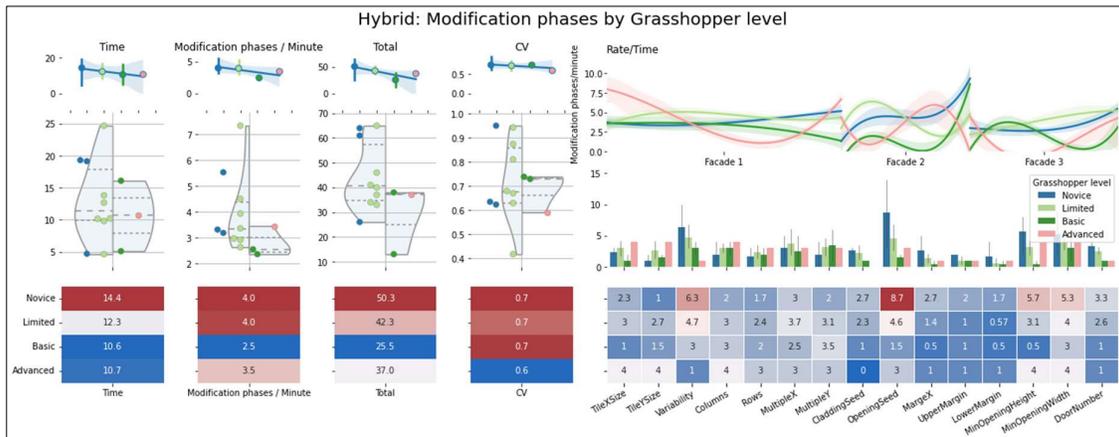
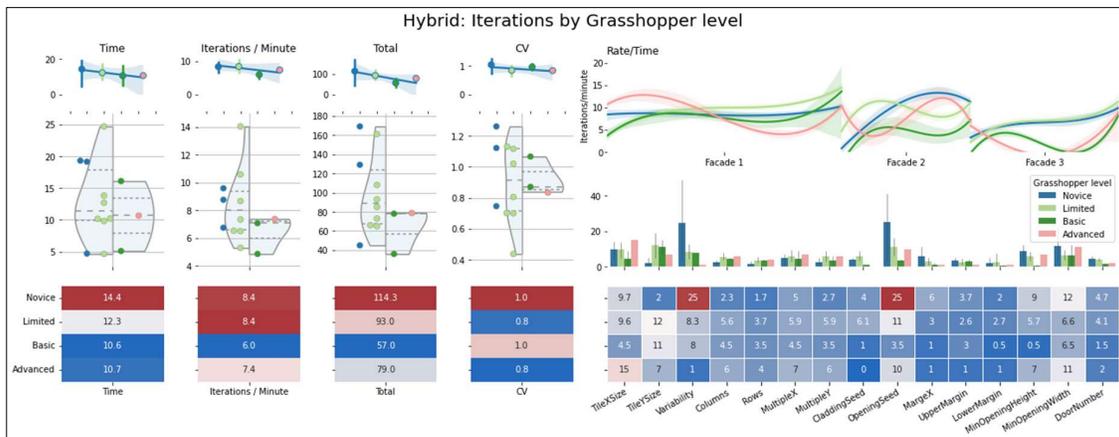
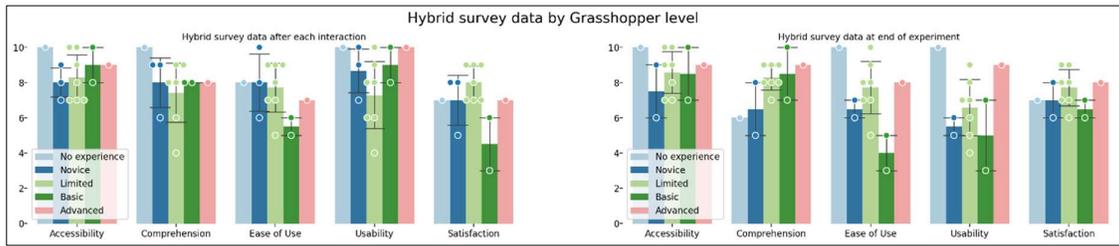
Appendix 21 : Hybrid by SketchUp



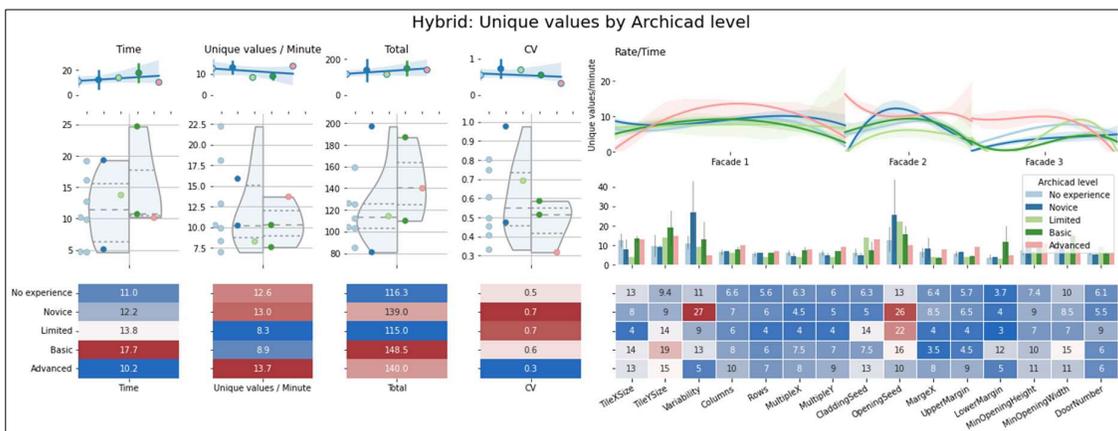
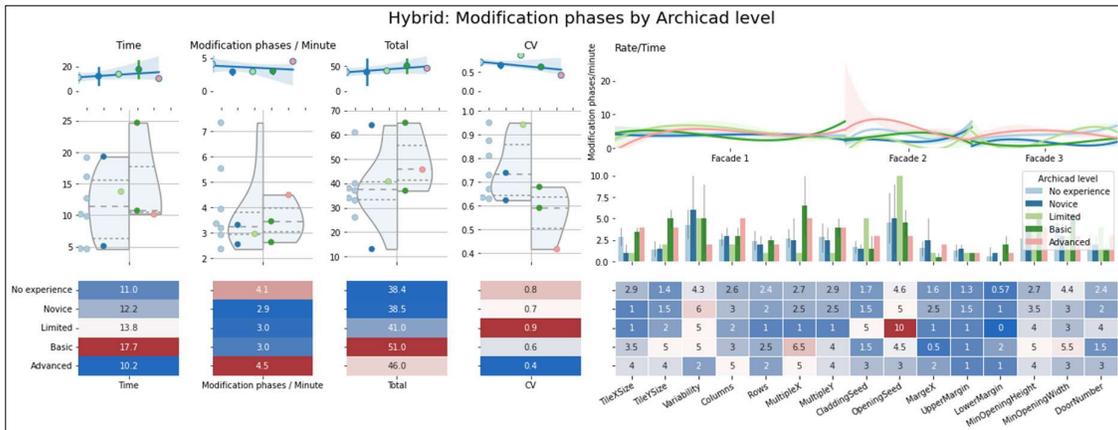
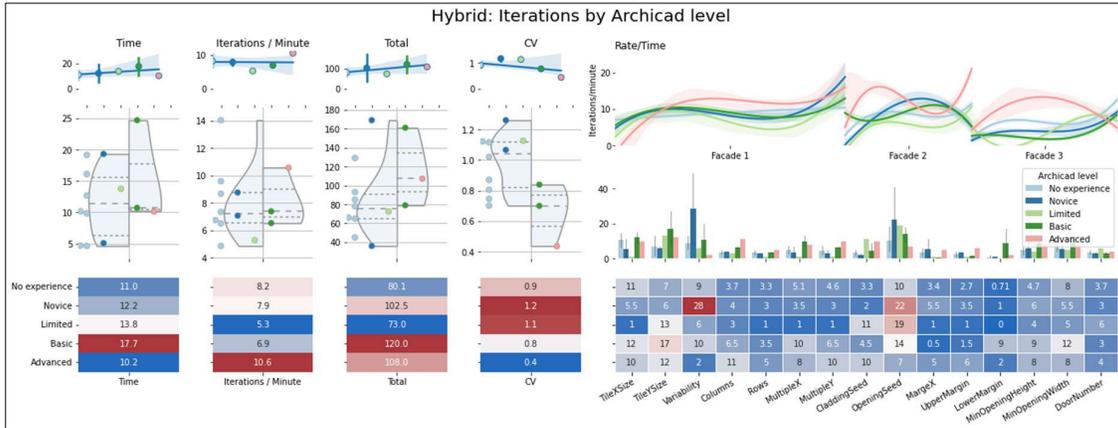
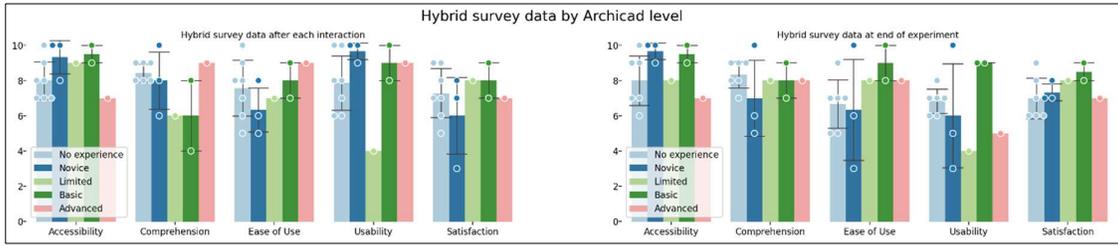
Appendix 22 : Hybrid by Rhino



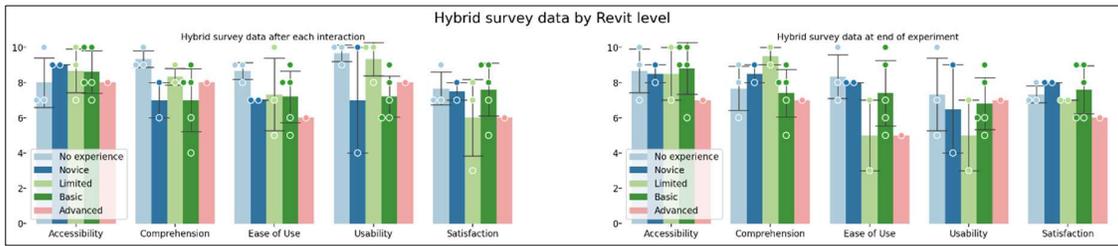
Appendix 23 : Hybrid by Grasshopper



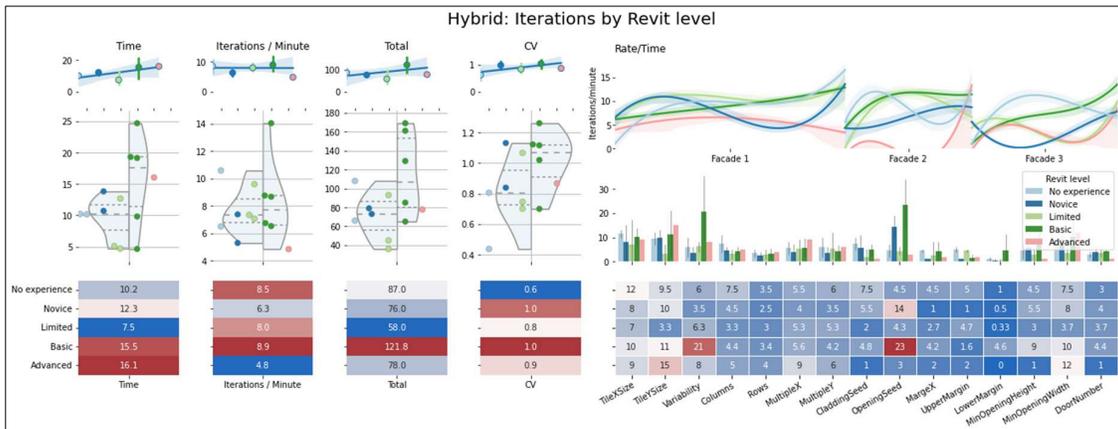
Appendix 24 : Hybrid by Archicad



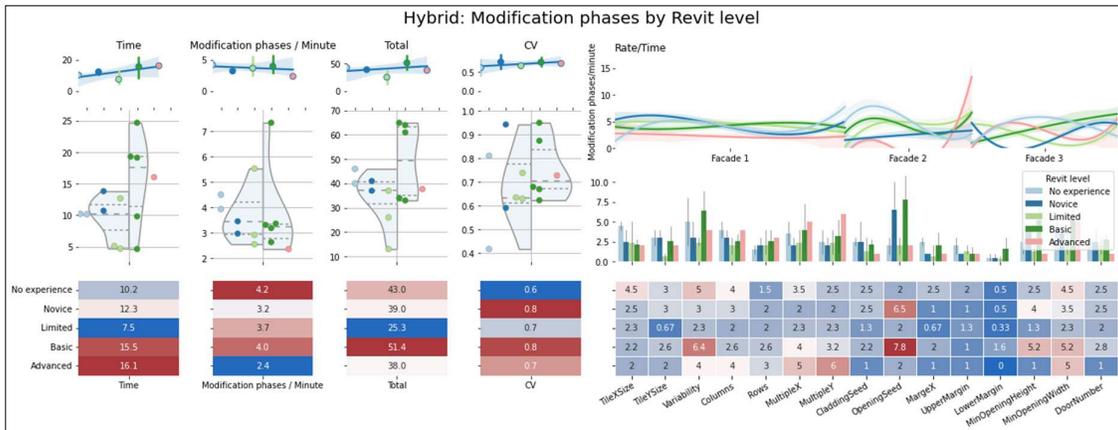
Appendix 25 : Hybrid by Revit



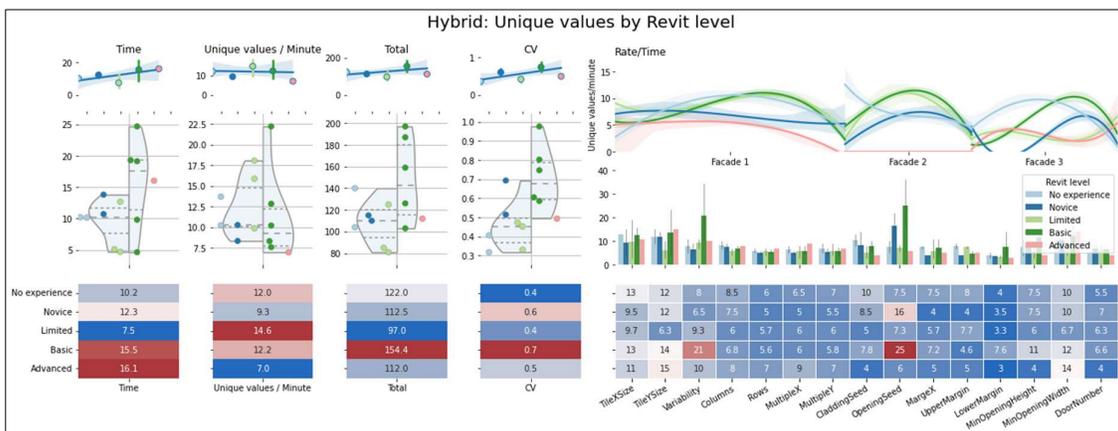
Appendix 25-1 : Hybrid survey data by Revit level



Appendix 25-2 : Iterations by Revit level

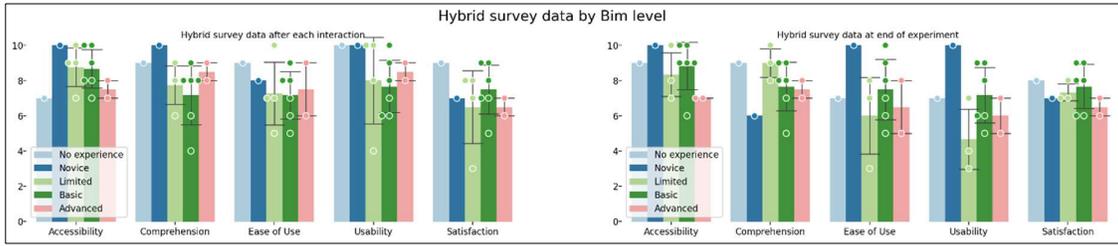


Appendix 25-3 : Hybrid-Modification phases by Revit level

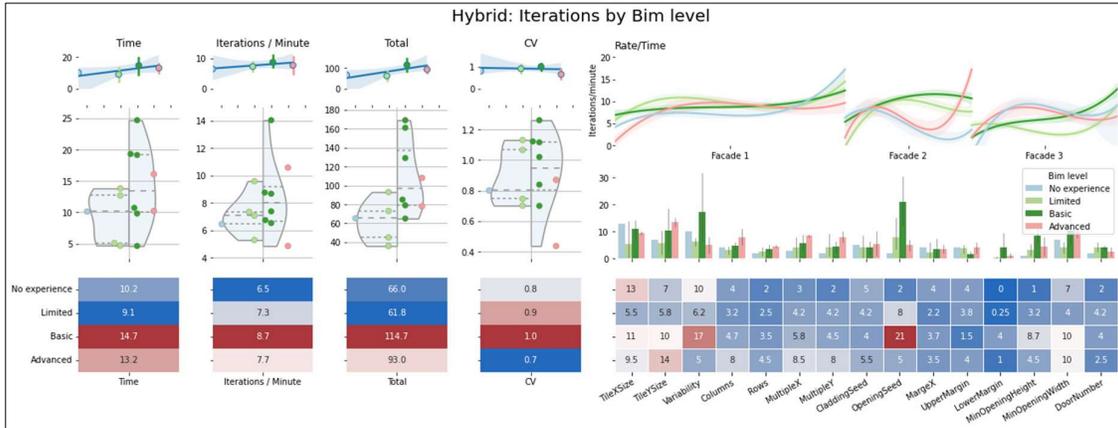


Appendix 25-4 : Hybrid-Unique values by Revit level

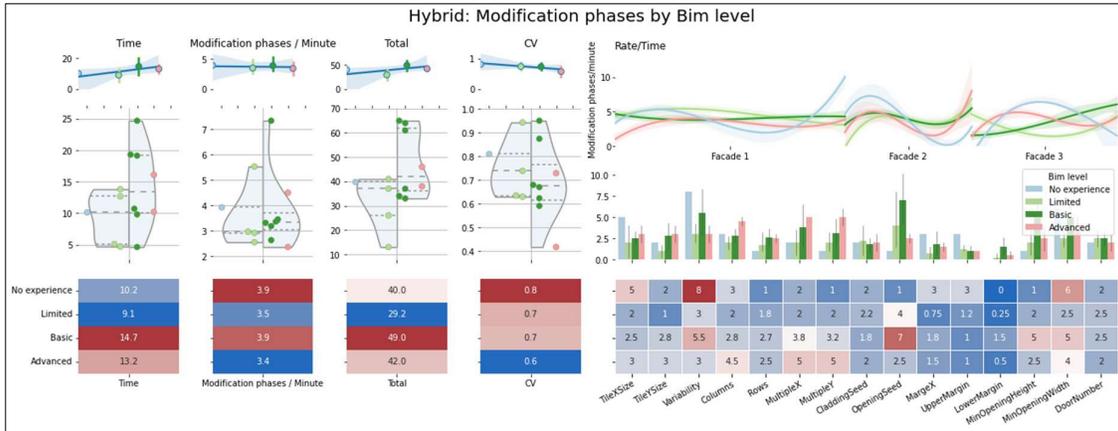
Appendix 26 : Hybrid by BIM



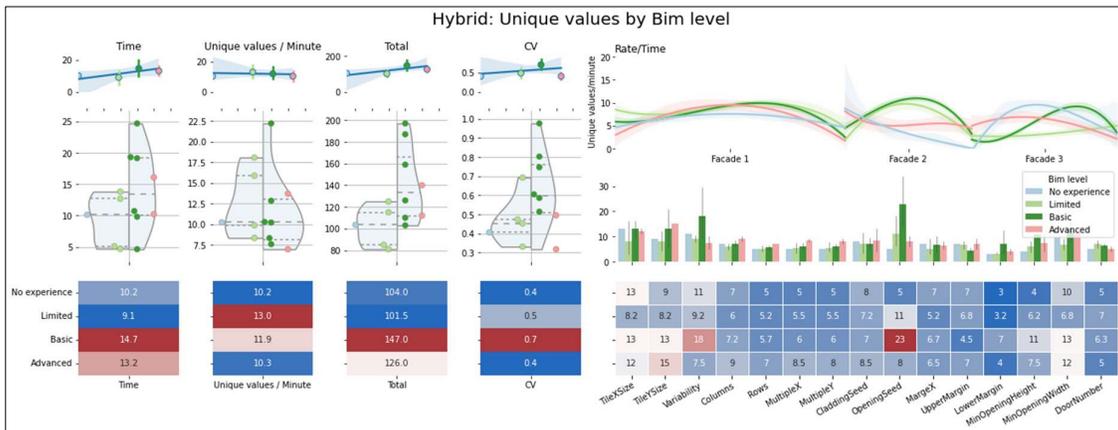
Appendix 26-1 : Hybrid survey data by BIM level



Appendix 26-2 : Hybrid-Iterations by BIM level

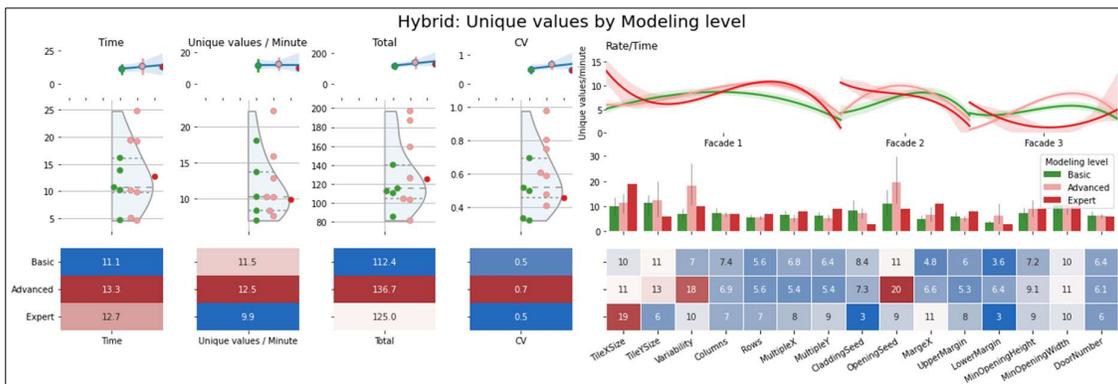
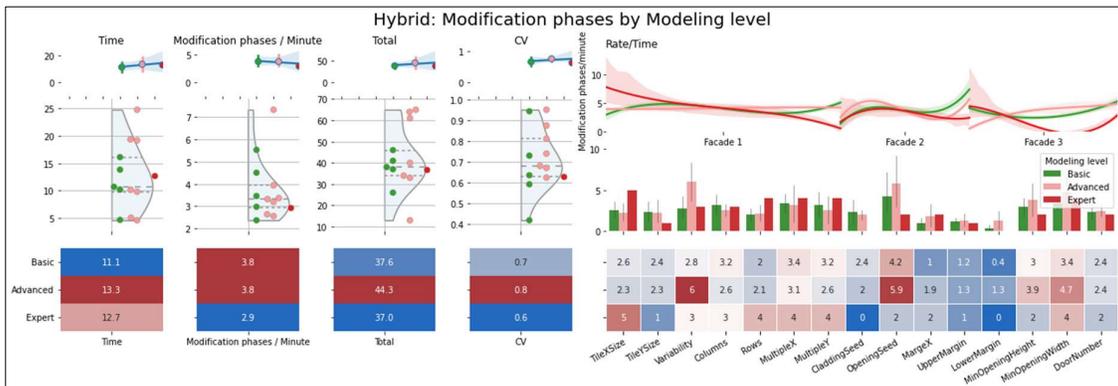
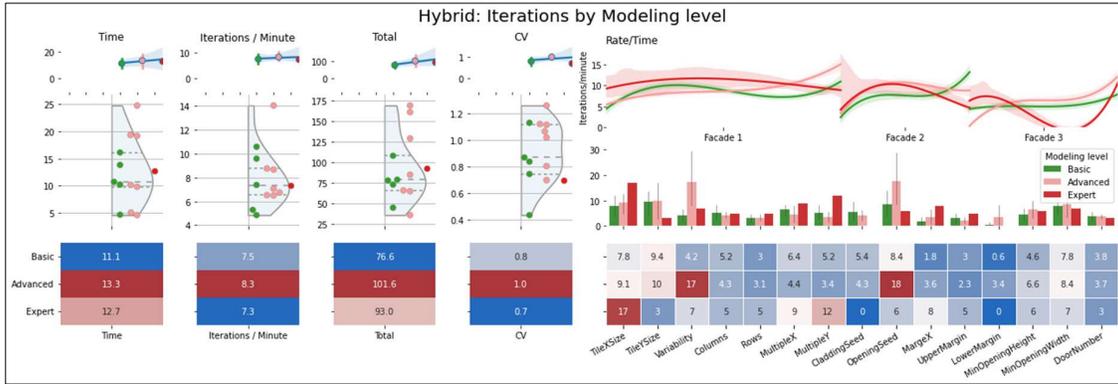
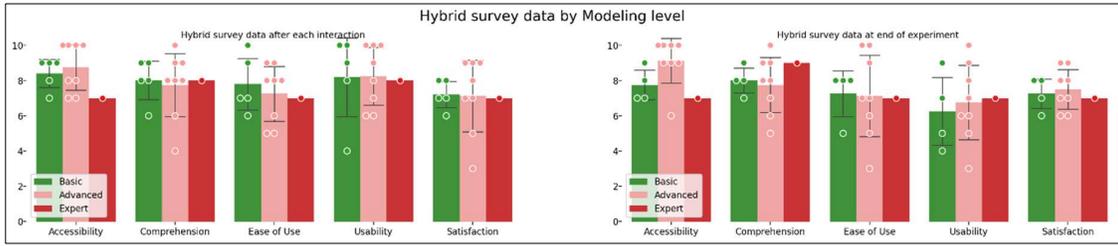


Appendix 26-3 : Hybrid-Modification phases by BIM level

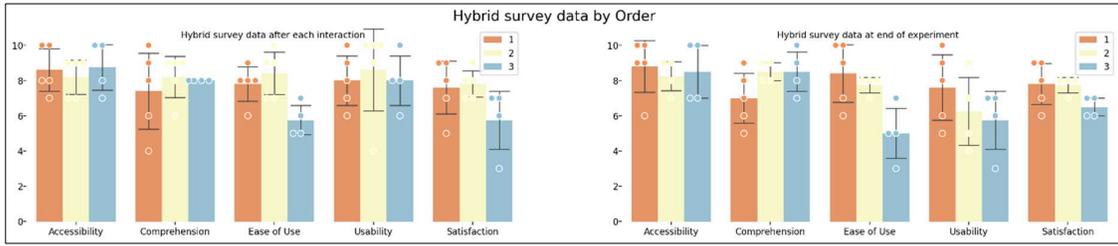


Appendix 26-4 : Hybrid-Unique values by BIM level

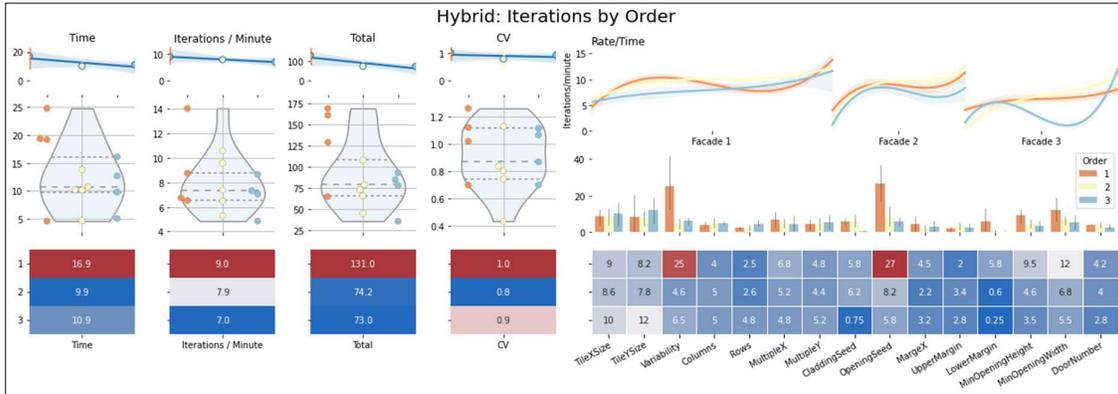
Appendix 27 : Hybrid by Modeling



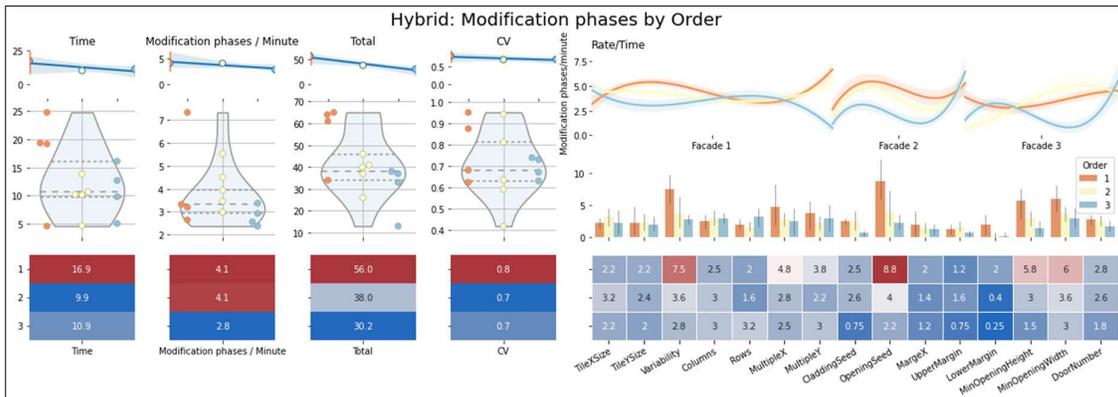
Appendix 28 : Hybrid by Order



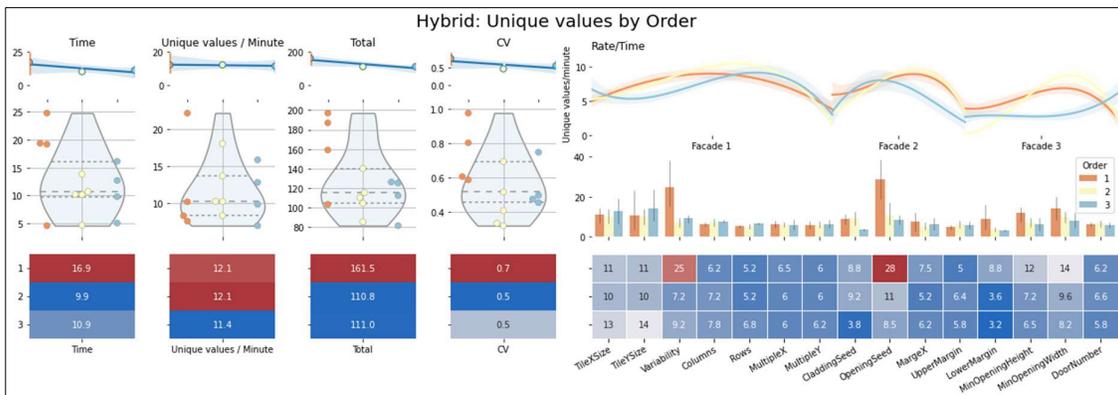
Appendix 28-1 : Hybrid survey data by order



Appendix 28-2 : Hybrid-Iterations by order

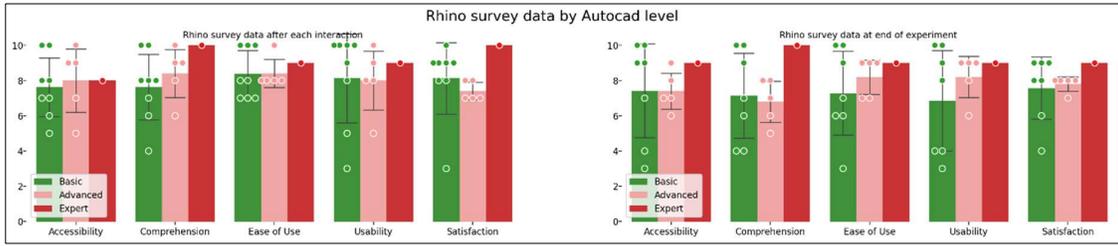


Appendix 28-3 : Hybrid-Modification phases by order

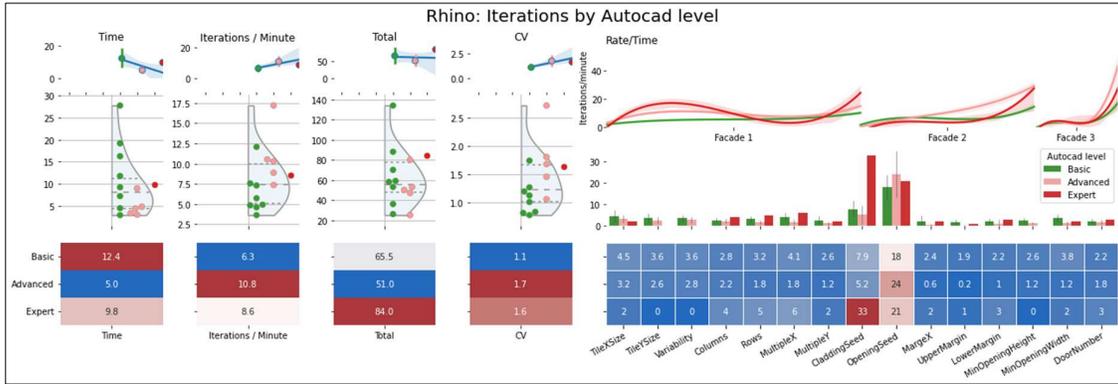


Appendix 28-4 : Hybrid-Unique values by order

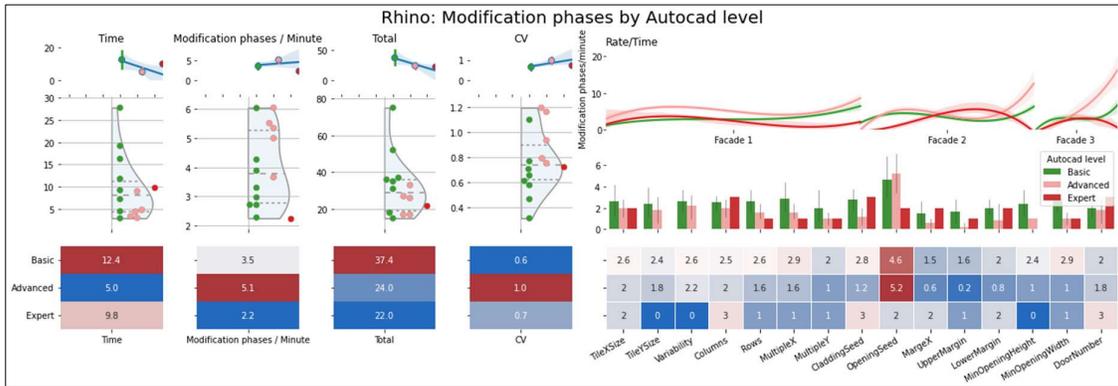
Appendix 29 : Rhino by AutoCAD



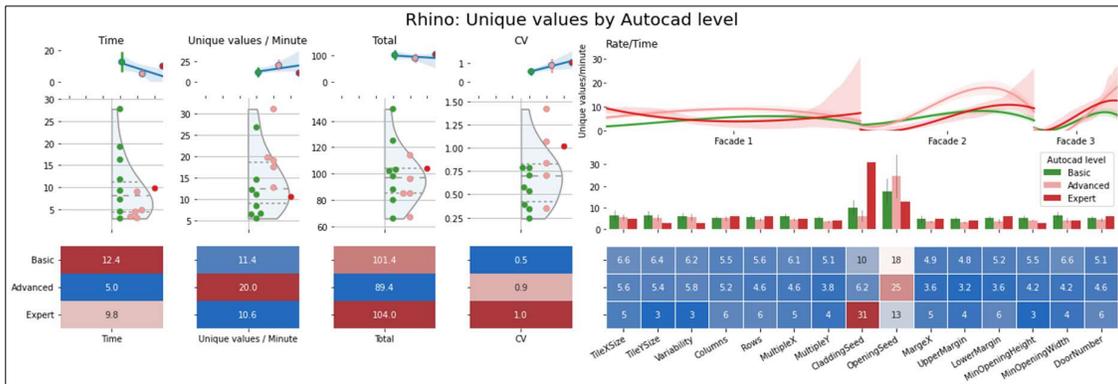
Appendix 29-1 : Rhino survey data by AutoCAD level



Appendix 29-2 : Rhino-Iterations by AutoCAD level

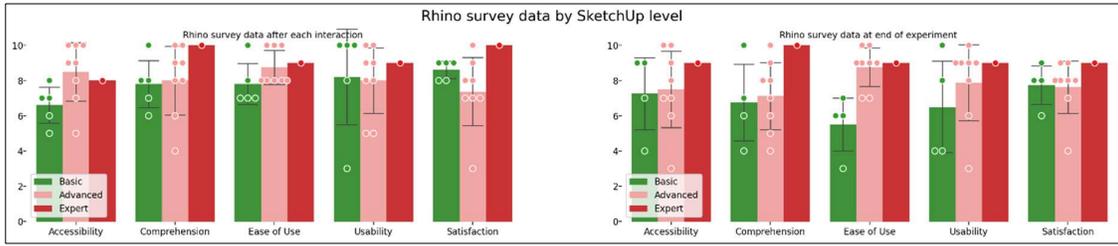


Appendix 29-3 : Rhino-Modification phases by AutoCAD level

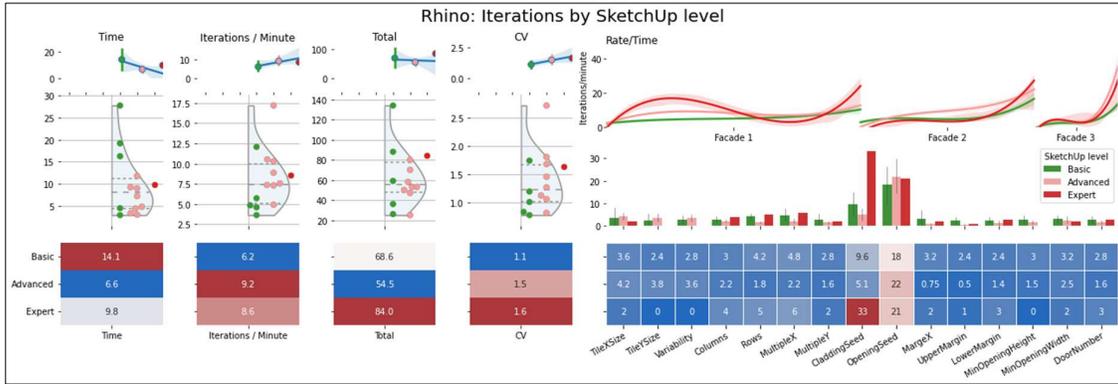


Appendix 29-4 : Rhino-Unique values by AutoCAD level

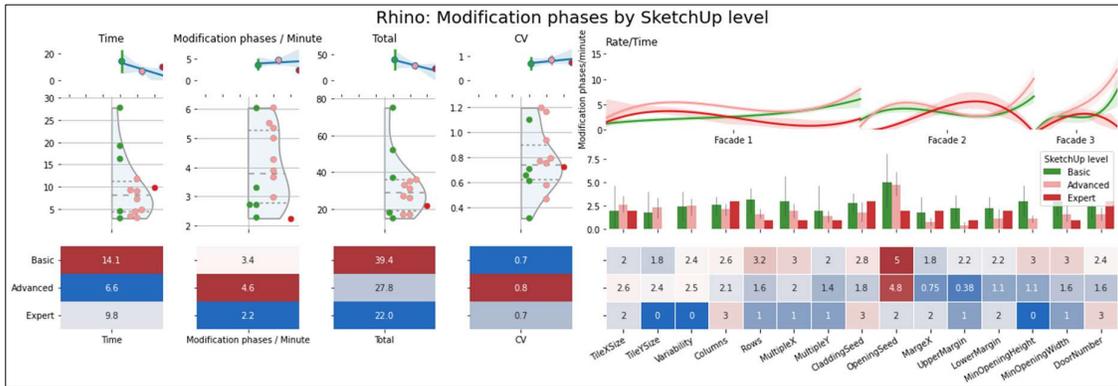
Appendix 30 : Rhino by SketchUp



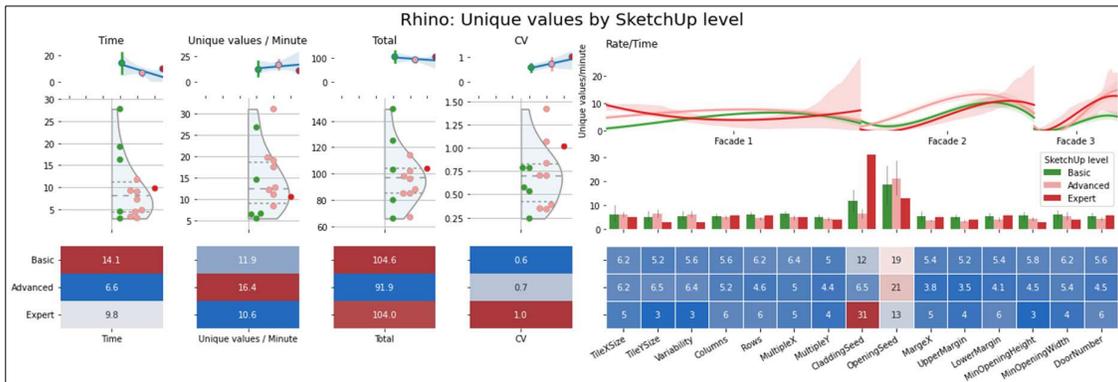
Appendix 30-1 : Rhino survey data by SketchUp level



Appendix 30-2 : Rhino-Iterations by SketchUp level

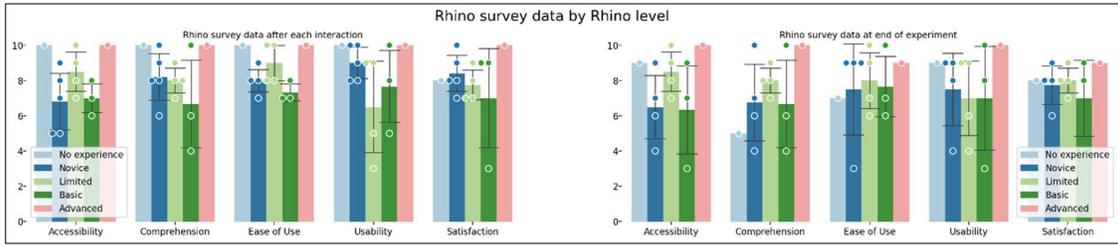


Appendix 30-3 : Rhino-Modification phases by SketchUp level

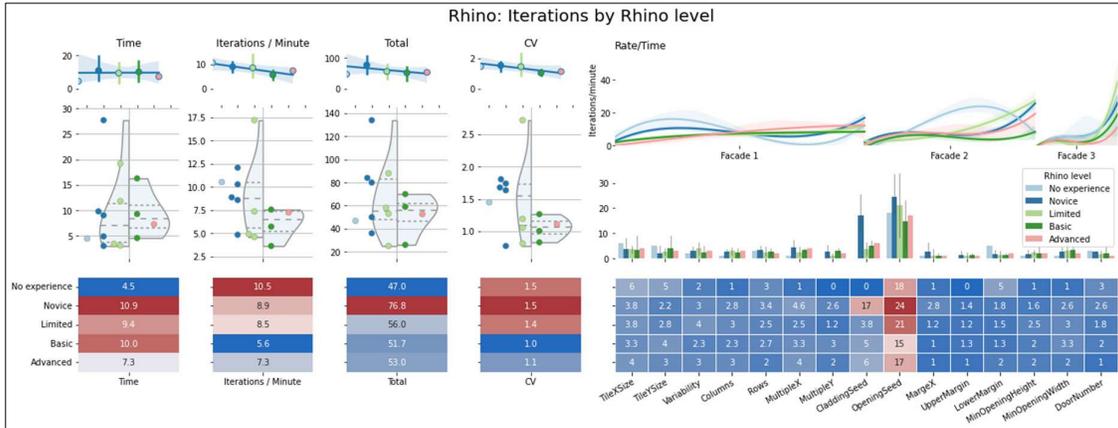


Appendix 30-4 : Rhino-Unique values by SketchUp level

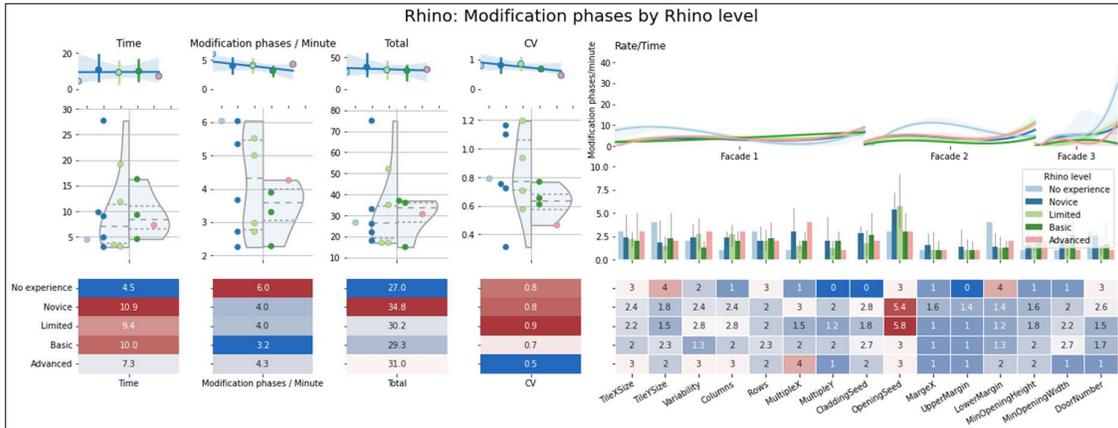
Appendix 31 : Rhino by Rhino



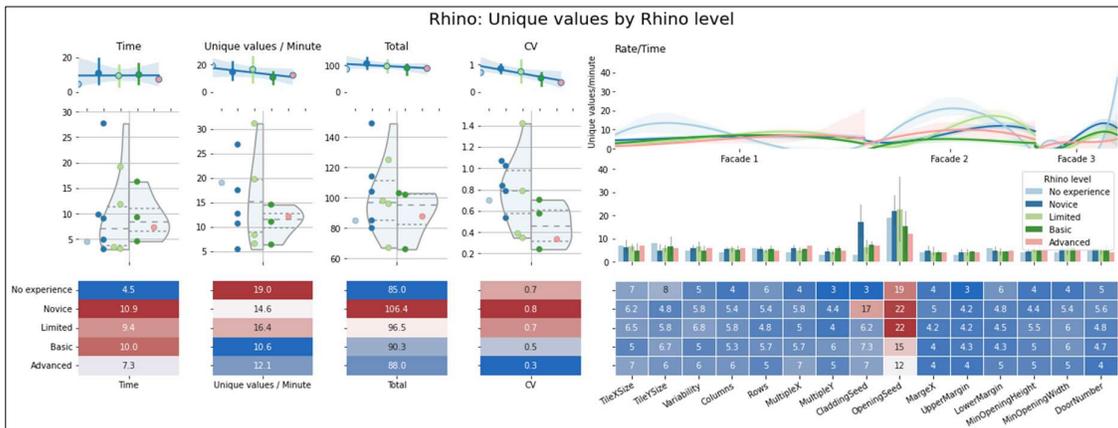
Appendix 31-1 : Rhino survey data by Rhino level



Appendix 31-2 : Rhino-Iterations by Rhino level

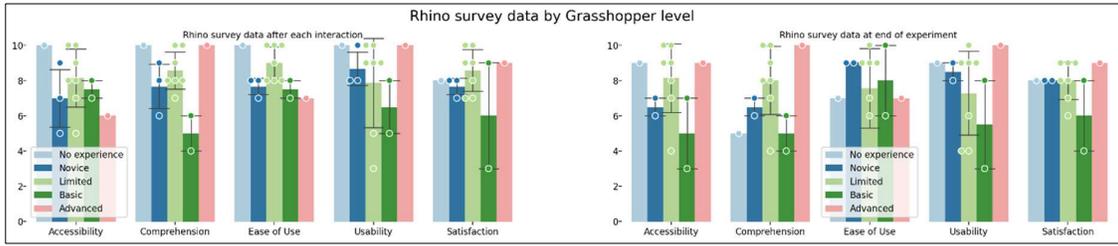


Appendix 31-3 : Rhino-Modification phases by Rhino level

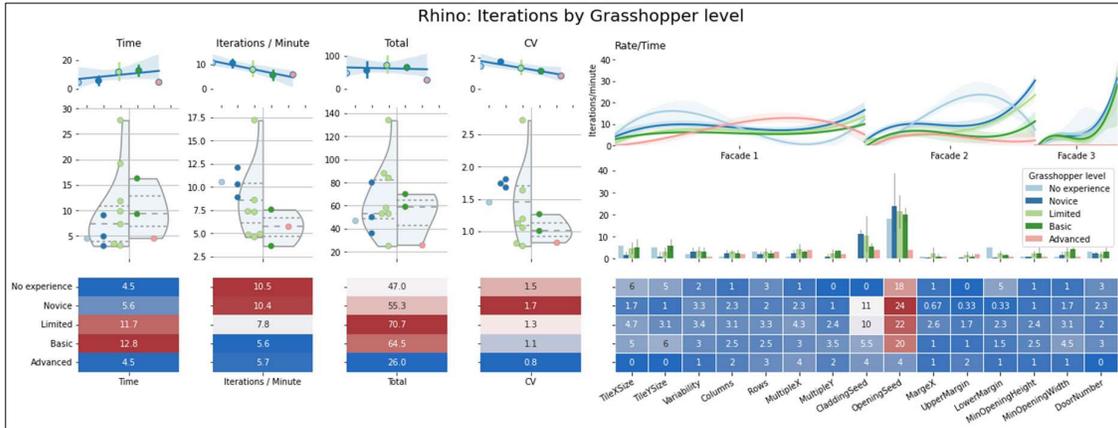


Appendix 31-4 : Rhino-Unique values by Rhino level

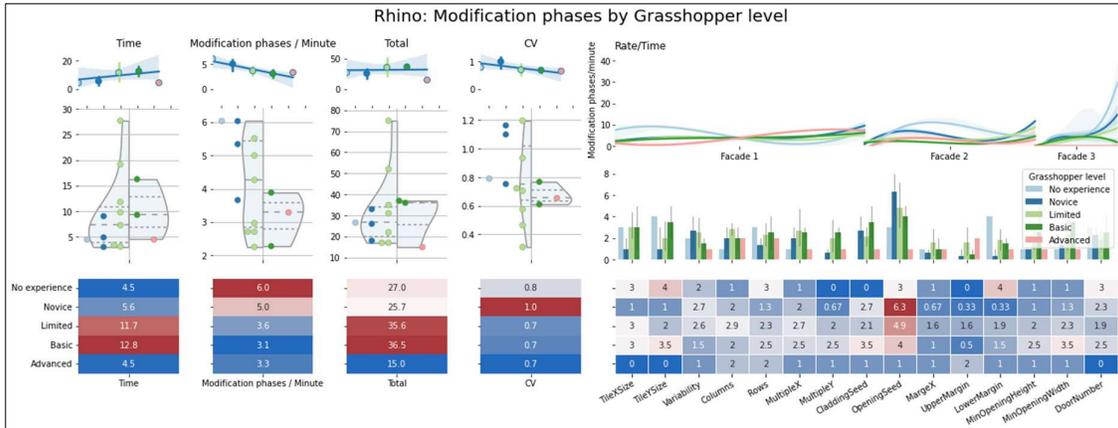
Appendix 32 : Rhino by Grasshopper



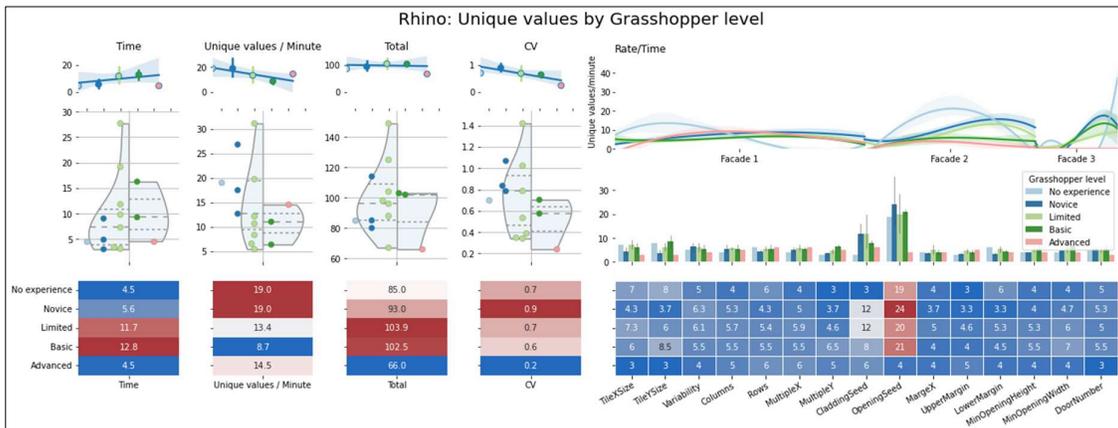
Appendix 32-1 : Rhino survey data by Grasshopper level



Appendix 32-2 : Rhino-Iterations by Grasshopper level

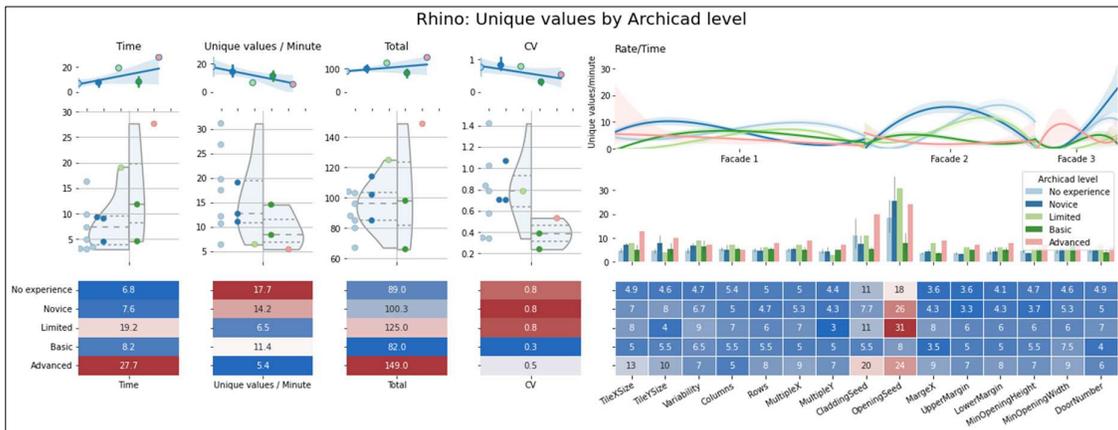
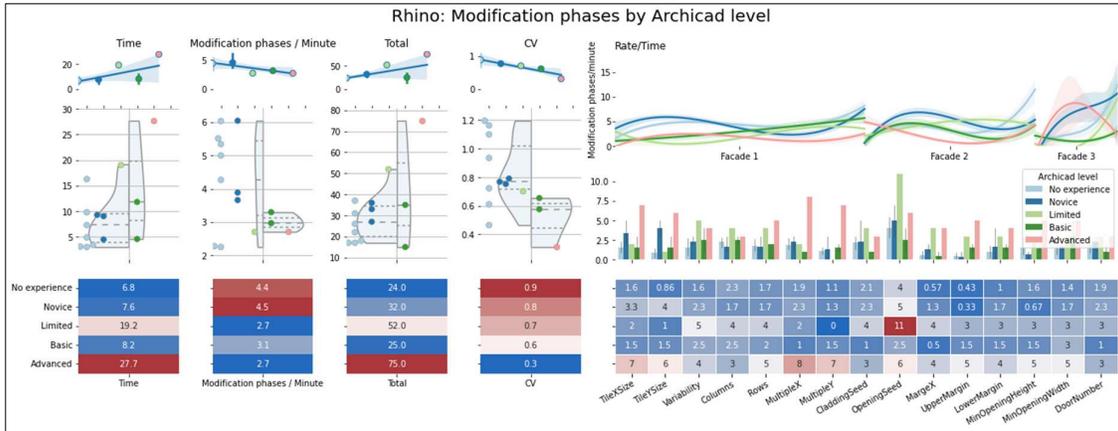
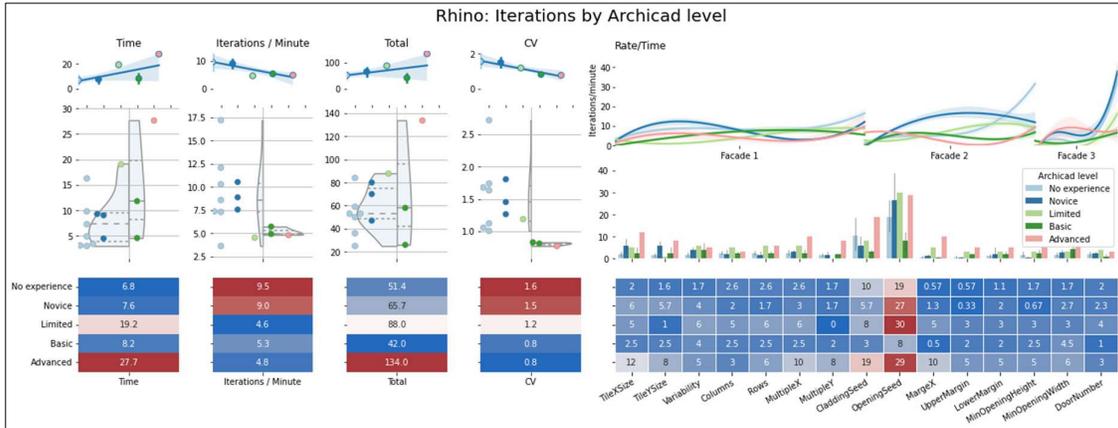
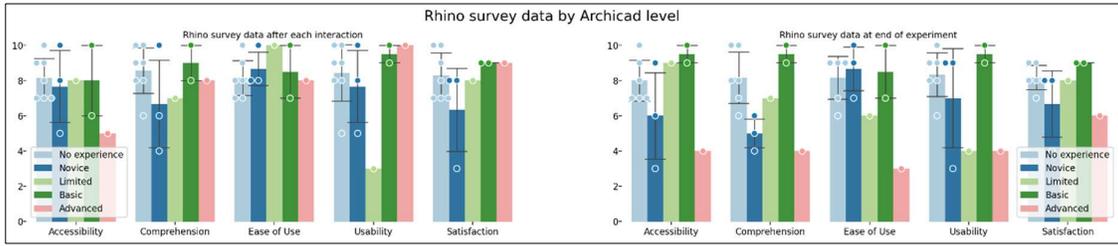


Appendix 32-3 : Rhino-Modification phases by Grasshopper level

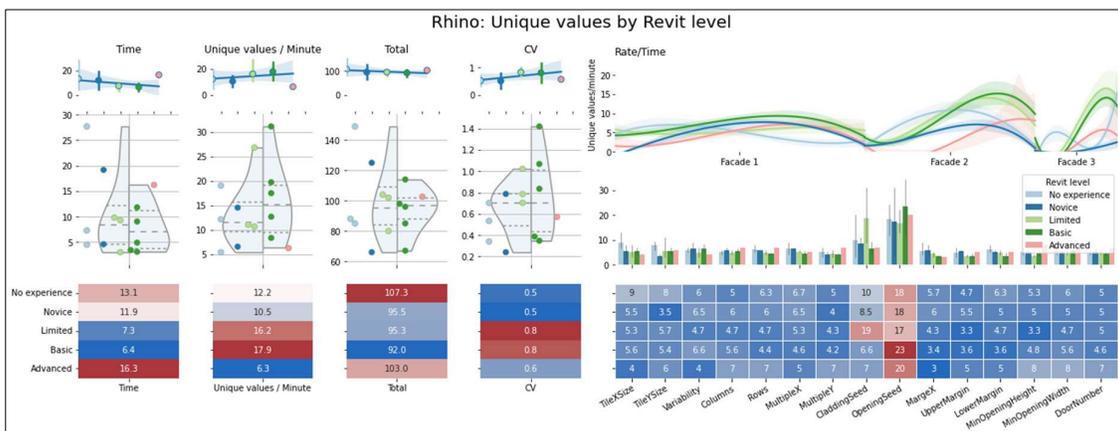
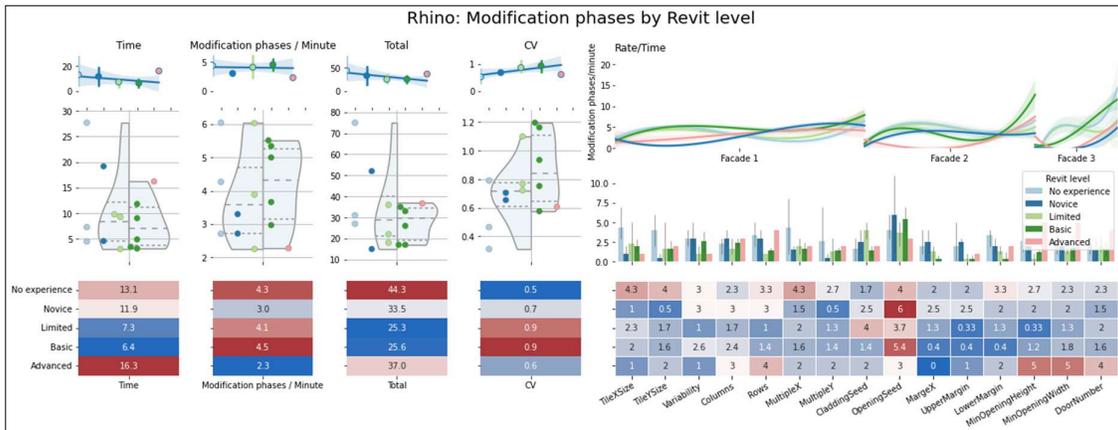
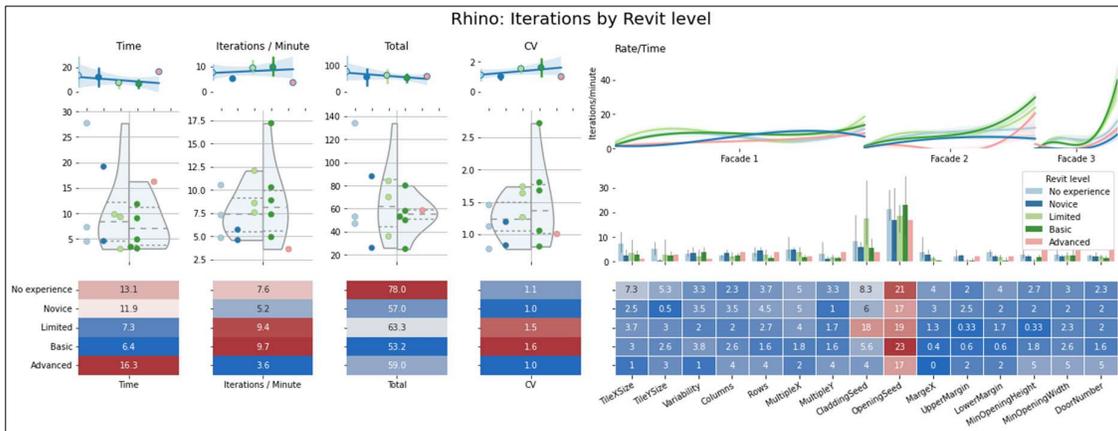
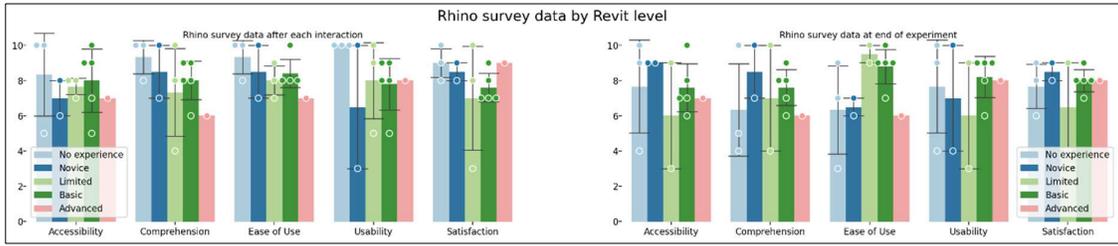


Appendix 32-4 : Rhino-Unique values by Grasshopper level

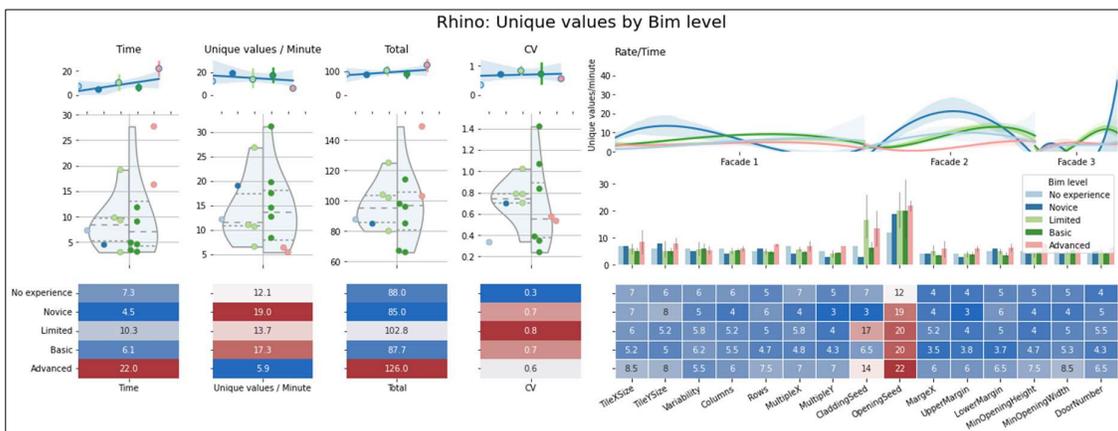
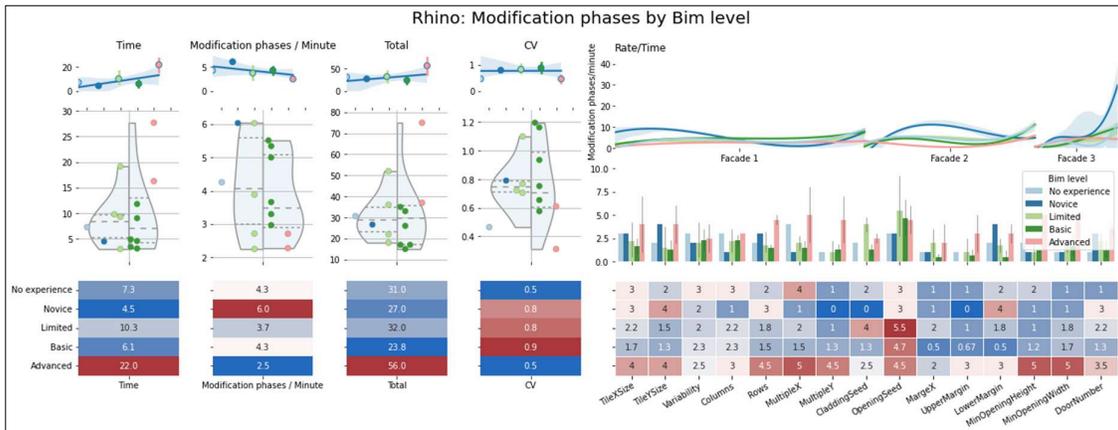
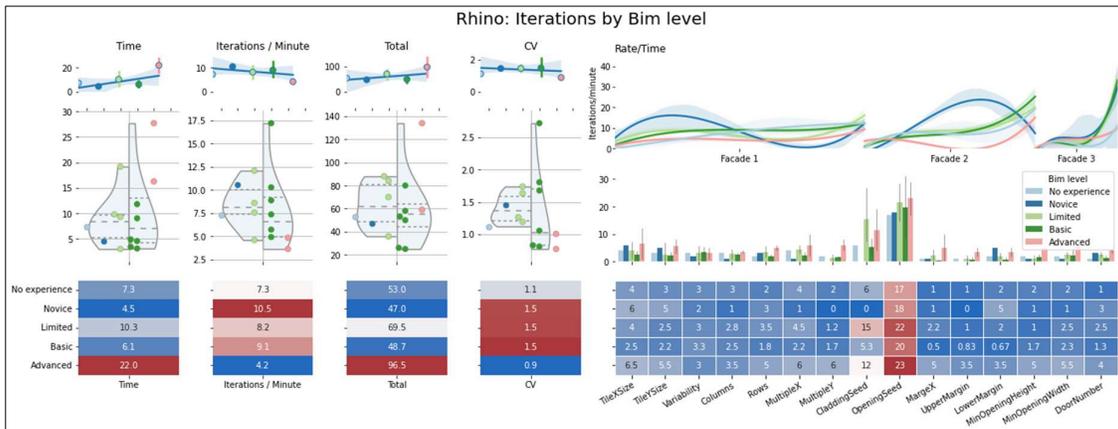
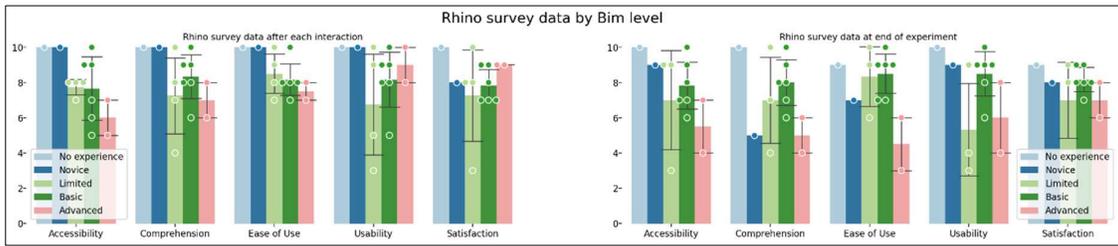
Appendix 33 : Rhino by Archicad



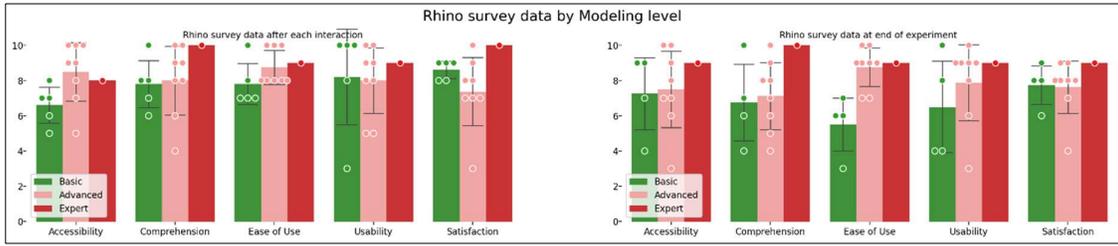
Appendix 34 : Rhino by Revit



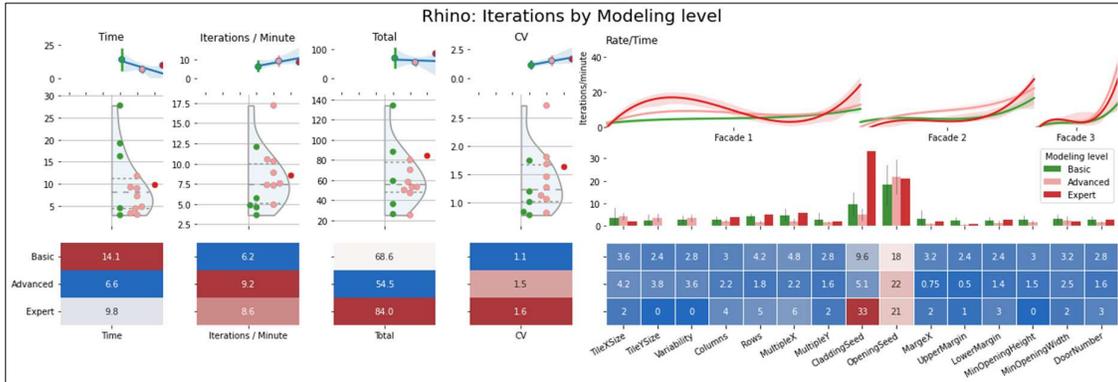
Appendix 35 : Rhino by BIM



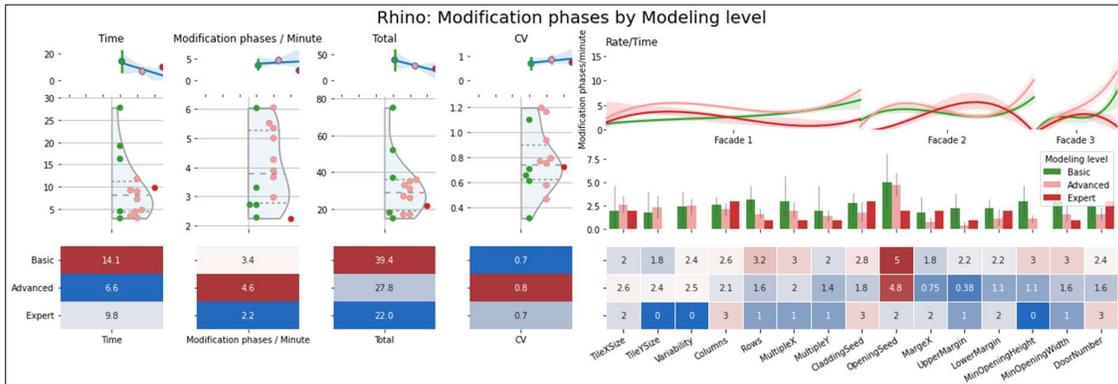
Appendix 36 : Rhino by Modeling



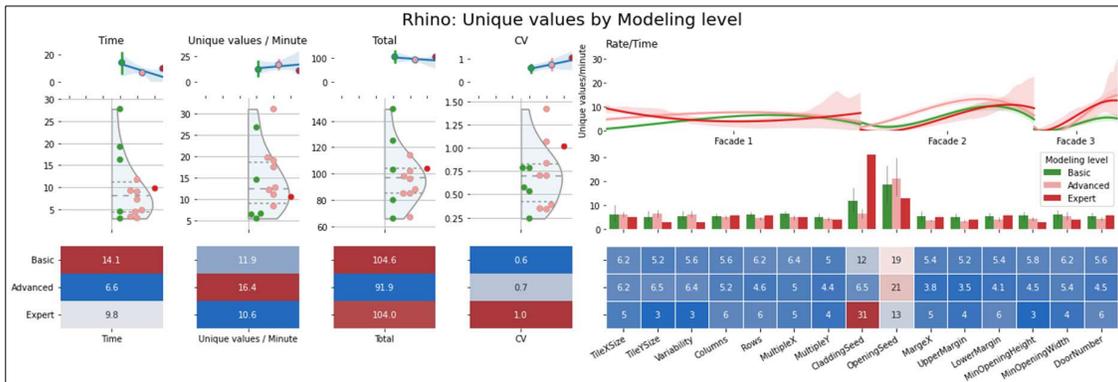
Appendix 36-1 : Rhino survey data by [Modeling level](#)



Appendix 36-2 : Rhino-Iterations by [Modeling level](#)

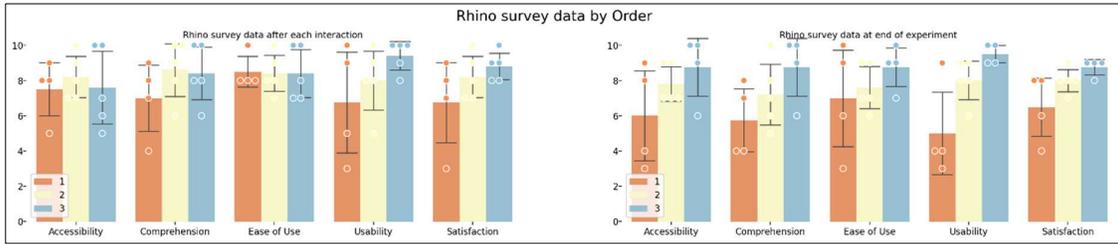


Appendix 36-3 : Rhino-Modification phases by [Modeling level](#)

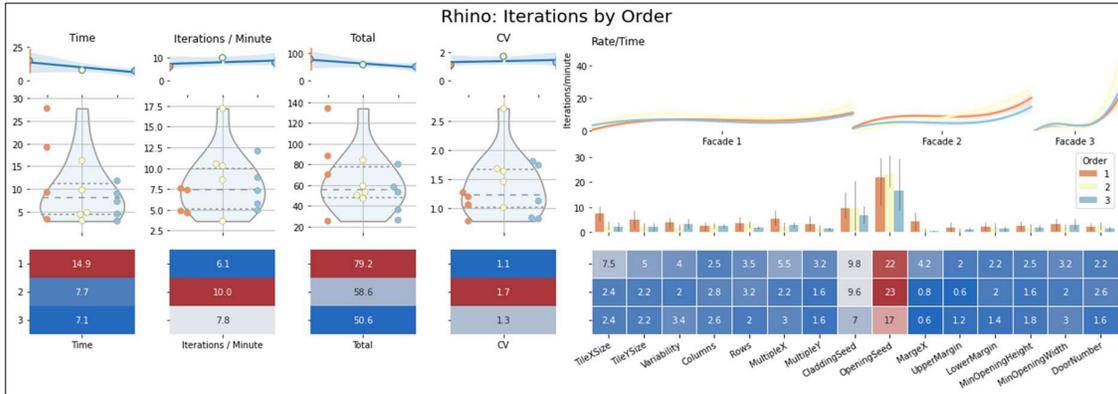


Appendix 36-4 : Rhino-Unique values by [Modeling level](#)

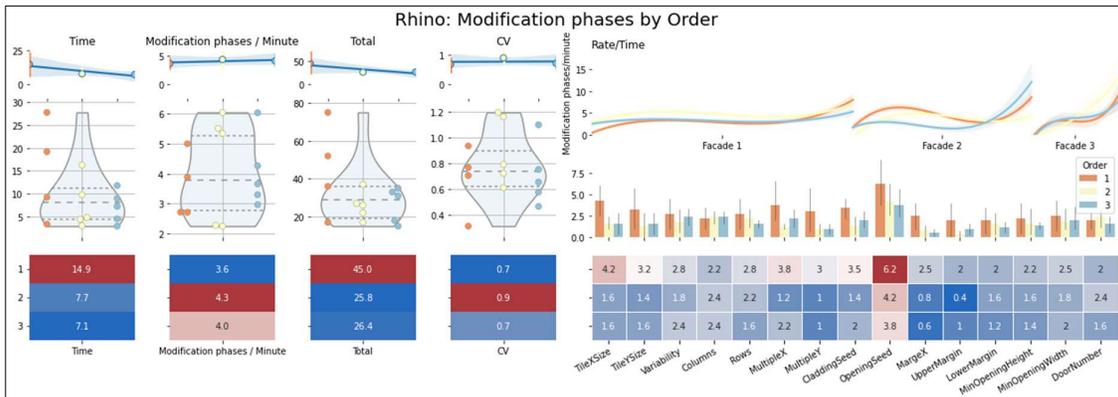
Appendix 37 : Rhino by Order



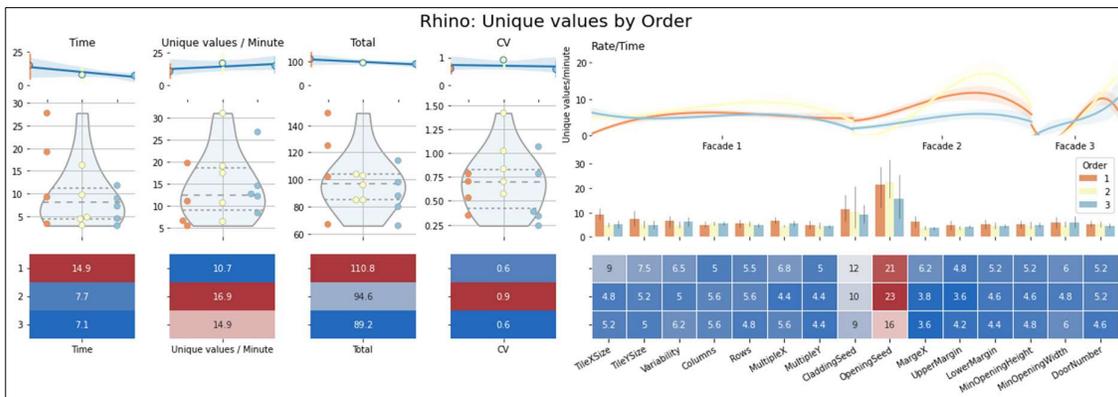
Appendix 37-1 : Rhino survey data by order



Appendix 37-2 : Rhino-Number of iterations by order



Appendix 37-3 : Rhino-Modifications phases by order



Appendix 37-4 : Rhino-Unique values by order