

CARNEGIE MELLON UNIVERSITY

School of Architecture

College of Fine Arts

Thesis

Submitted in Partial Fulfillment of the requirements for the degree of

Master of Science in Computational Design

TITLE:

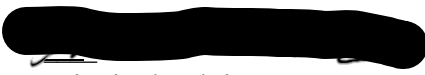
**Generative Design Software Development Process for
Architecture**

AUTHOR:

Siyu Guo

ACCEPTED BY ADVISORY COMMITTEE:

Daniel Cardoso Llach


Principal Advisor

August 7, 2020
DATE

Generative Design Software Development Process for Architecture

By Siyu Guo

*B.S. in Architecture -- Illinois Institute of Technology
Master of Science in Computational Design
at the
Carnegie Mellon University*

*Thesis Supervisor -- Daniel Cardoso Llach
Associate Professor of Department of Architecture
Track Chair of Computational Design*

Abstract

The existing design method of using traditional CAD (Computer Aided Design) software to explore architectural design alternatives is time-consuming and tedious. Generative Design (hereinafter called the GD) software is a potential solution that can streamline the design process and provide creative design alternatives for architects. However, current GD software is not widely accepted by architects because they are either difficult to use, or their generated results don't meet architect's needs because the solutions it generates don't always meet design criteria. This situation reflects a more fundamental reason that limits the promotion of GD software which the industry lacks a software development process that could guide developers to develop valuable GD software for architects.

Therefore, in this thesis, I propose a development process to fill the gap. The process consists of four major steps and it integrates the necessary architectural design knowledge and programming resources to follow and reference. To demonstrate the feasibility of the process, I developed a GD software for a basement parking layout design task. Based on the generative method of *variational modeling with optimization*, this tool can generate a variety of valuable basement parking layouts in a very limited amount of time. According to the software test result, the generated solution reaches about 86.95% accuracy but reduces 36% modeling time when architects use it to reproduce an existing basement parking layout. The GD software developed under the guidance of the process proved to be valuable.

Acknowledgements

I am sincerely thankful to the following individuals for their encouragement, support and Inspiration.

Prof. Daniel Cardoso Llach for cultivating me to be a researcher, for introducing me to the realm of Computational Design, for teaching me how to conduct research, and countless of great insights and useful suggestions to my research.

The member of the Code Lab, Vincent Mai, Hongtao Ma, Zhihao Fang, Weixin Qiu, Yun Hao, Yufei Cheng, Yichin Lee, Yixiao Fu, Yaxin Hu, Ian Friedman, Erik Ulberg for many hours of great discussion.

The researchers from Autodesk, Aaron Lu, Andy Du, Christopher Zhang for brainstorming and technical support.

My family and friends.

TABLE OF CONTENTS

Chapter 1. Introduction	6
Chapter 2. Background	9
2.1 Rule-based modeling	10
2.2 Rule-based modeling with search	12
2.3 Variational modeling with optimization	18
2.4 Simulation-based system	21
2.5 Reflection on the case studies	25
Chapter 3. Hypothesis: A GD software development process	28
3.1 Identify appropriate design task to develop GD software	30
3.2 Ideate the most suitable generative method for selected task	31
3.3 Develop prototype with visual programming platform	32
3.4 Test prototype's performance with customized dimensions	33
Chapter 4. Methods	34
4.1 Select basement parking layout design task	35
4.2 Select generative method for basement parking layout design task	36
4.3 Develop prototype based on variational modeling with optimization	41
4.4 Test the prototype with practical basement projects	46
Chapter 5. Discussion	49
5.1 Contribution	49
5.2 Limitation	49
5.3 Next Step	50

Chapter 1. Introduction

The existing design method of using traditional CAD software to explore and model architectural design alternatives is time-consuming and tedious. Generative design software is a potential solution that can streamline the design process and provide potential creative design alternatives for architects to consider. Below is my definition:

“Generative Design is a collaborative design process between architects and generative software. Architects input modules, parameters, rules, constraints for specific design tasks, the generative design software uses specific computational approaches to automatically generate, optimize and rank the design solutions for architects.”

In the past 40 years, plenty of CAD software companies, academic institutions and individual developers participated in the research of GD software development, however, after decades of development, GD software is still not widely accepted by the current architectural design industry. There are two major reasons that the promotion of this new type of design software was limited: They are difficult to use, and the quality of the generated design alternatives doesn't meet architect's needs (Davis Davis 2020). Take Galapagos as the first example. Developed by David Rutten, Galapagos is a GD plugin that provides evolutionary algorithms for any architectural design problems (Rutten 2010). But, architects reported that the program required a high level of programming and algorithm knowledge in order to be useful. Another good example is the Xkool. Published in 2016, the team of Xkool claimed that this GD software meant to provide solutions for building site layout design. However, users complained that the system took too long to generate design alternatives due to its bad computational approach. Besides, some generated alternatives turned out to be useless because the system missed necessary building design criteria and constraints.

As evidenced above, developing a valuable GD software is not easy work, it requires the developers to have in-depth knowledge on both architectural design and computer engineering. It is unrealistic to ask all developers to gain expertise on both sides before they develop a GD software, rather, the industry requires a GD software development process that integrates all the necessary knowledge and provides a standardized development process for developers to follow.

Back in the 90s, there was a recognized GD software development process. Proposed by Robert F Coyne and revised by Ulrich Flemming, the goal of this development process was to guide developers from different backgrounds to efficiently develop valuable GD software, especially the GD software for architectural layout design problems (F Robert 1993; Flemming 1994). Layout design was deemed more important than others because layout design permeates building design at most of its stages and thus provides a rich context for the definition and investigation of problem domains with various degrees of complexity (Flemming, Coyne, and Glavin, 1995). However, I found out this process was not suitable for today's industry environment due to the following limitations:

- 1) It doesn't provide enough information regarding how to select suitable design tasks for developing a GD software.
- 2) It doesn't provide a guideline to select appropriate generative methods for different design tasks, this process is based on an unique generative method.
- 3) It doesn't provide methods regarding how to efficiently acquire necessary architectural design knowledge for selected design tasks. In addition, It's developed on the outdated development platform.
- 4) It's missing a prototype test process.

Therefore, the goal of this thesis is to explore a new GD software development process that addresses these shortcomings. I raised my research questions as such: ***What is the correct process for developing a GD software suitable for architectural layout design problems under the current background of architectural design and CAD software industry?***

To find the answer, I started my exploration with multiple carefully selected GD softwares and prototypes for case studies. By comparing and analyzing their differences in application scenarios, generative methods, software architecture and development platforms, I've summarized the following major findings:

- 1) The development of GD software should begin with identifying whether a design task is suitable for developing GD software. The GD software tends to solve design problems that: the problem related design goals are quantifiable, and the current method of using CAD software to explore design alternatives for this problem is low efficient.

- 2) Currently there are three major generative methods that developers used for building GD software to solve layout design problems. They are *rule-based modeling with search*, *variational modeling with optimization*, and *simulation-based systems*. Each of them is suitable for different usage needs.
- 3) Currently the best platform to develop a GD prototype is either dynamo or grasshopper.
- 4) A customized GD prototype testing process for different design tasks is always needed.

In chapter three, I proposed a new GD software development process based on the findings I summarized in previous chapters. The new process consisted of four major steps: Step one, Identify the appropriate design task to apply GD technology. Step two, Ideate the most suitable generative method to build the GD software for the selected design task. Step three, develop prototypes with visual programming platforms. Step four, test prototype's performance with customized dimensions. I explained how the developer should implement each step in detail.

In chapter four, in order to demonstrate the feasibility of the development process, I developed two GD software for a basement parking layout design task. With the variational modeling and optimization approach, this GD software can generate a variety of parking stall's layout in a very limited amount of time and is capable of maximizing the total number of stalls for each design alternative. After the prototype tested with two predefined dimensions, it proved to be valuable.

Finally, in chapter five, I summarized the contributions and limitations of this new development process. In all, the proposed process is good to be used as a guidebook for developers who are willing to develop GD software for layout design tasks. It provides detailed explanations regarding how to efficiently pick the right generative method, acquire necessary architectural design knowledge, and develop prototypes with existing resources. Meanwhile, it provides a dynamo code template for developers to refer to. Furthermore, the process could inspire AEC practitioners to rethink what is a good GD software in the field of architectural design and how we should develop it. eventually it will accelerate the industry to widely accept generative design tools as part of their daily software toolbox.

Chapter 2. Background

GD software evolves with the change of its underlying generative methods. Generative methods mainly refer to the combination of algorithms and rules that a GD software uses to generate design results. In the past few decades, developers have continuously introduced the latest generative methods into the architectural design industry to solve different types of design problems. Among them, there are three major generative methods used to solve layout design problems: rule-based modelling with search, which is based on shape grammar; variational modeling with optimization, which is based on parametric design and evolutionary solver; simulation-based system, which is based on cellular automata algorithms (Krishnamurti and Veloso 2019).

In this chapter, I studied the logic, application scenarios, usage conditions, and related representative GD software for each of these three methods in chronological order. Meanwhile, I carefully analyzed a GD software development process that purely relies on the rule-based modeling and pointed out the shortcomings of developing GD software with a single generative method. Then I proposed the idea of selecting suitable generative methods to develop GD software for different design tasks. Finally, I summarized how to improve and complement the existing process based on the new idea so that the process will better serve today's developers.

2.1 Rule-based modeling

Among the three methods introduced above, the Rule-based modeling is the earliest method that was used to solve the architectural layout design problem. In general, it is a design method that uses a set of shape rules, selection rules, shape state, and algorithmic logic to generate 2D design configurations. The concept originated from George Stiny's shape grammar (George Stiny and Gips. 1972) and it was introduced by CAD researchers to the architectural design industry as a new method to explore the possibilities of architectural layout. These researchers believe that architects could use a set of predefined shape rules and selection rules which represent the characteristics of a classical architectural style to explore new possibilities of layout configurations in that style. Besides, this method could also assist architects to identify whether an existing floor plan belongs to the style.

To test the feasibility of the idea, these researchers developed several tools targets on different classical architectural style such as Palladian grammar (G Stiny and Mitchell 1978), the Mughul Gardens gramm (G Stiny and Mitchell 1980), and the Prairie House grammar (Koning and Eizenberg 1981). Among them, one of the most popular tools is the Palladian grammar.

Palladian Grammar

Developed as a definition of the Palladian architectural style, the Palladian Grammar is capable of generating the ground floor plans of Palladio's villas by its shape rules and generative engine (G Stiny and Mitchell 1978). To create a complete Palladio floor plan by the grammar, users need go through 8 design phases including: grid definition, exterior-wall definition, room layout, interior-wall realignment, principle entrance, exterior ornamentation, windows and doors, and termination. There is a shape-state to check whether the phase is complete, only when the current phase is judged to be completed, the user can go to the next phase.

Under each phase, there are 69 different shape-rules in total. A typical shape rule consists of 2 parts: geometry on the left-hand side which indicates the shapes that can be recognized and replaced by this rule, and geometry on the right-hand side, which indicates the shape that is used to replace the left-hand side shape. The process of applying shape rules to the existing design space is essentially a shape replacement process. As fig 1 shows, the user picked the shape-rule

number 12 from phase 3 (room layout phase), he found out this rule could be applied to the center of the existing floor plan since there is a ‘three connected cell’ area that matches the left-hand side information of the rule 12. Then he placed this rule 12 in this area, the layout of the existing floor plan was changed.

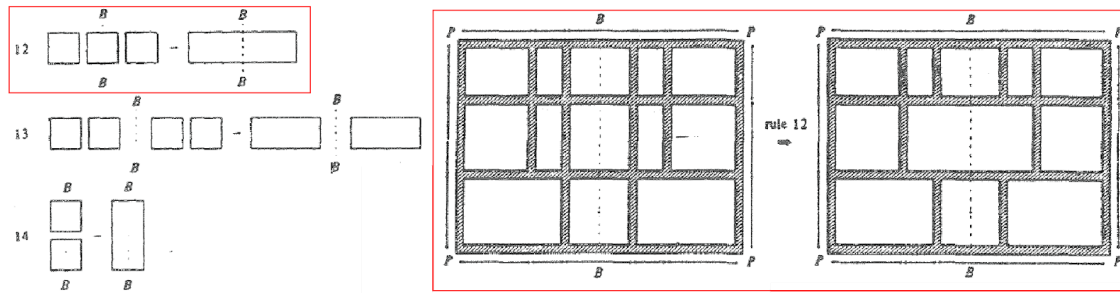


Figure 1. Palladian Grammar, Phase 3, Apply rule 12 to change the villa floor plan.

Through using the method provided by Palladian Grammar to create and identify villa plans, architects can explore many creative layout figurations that are hard to be found through the traditional method. The Palladian Grammar was not a software at the time, it was a manual design tool that developed based on a generative method. However, the underlying logic of this tool has given it the potential to become a GD software. Several years later, it was developed as web-based GD software by the team GRAPP.

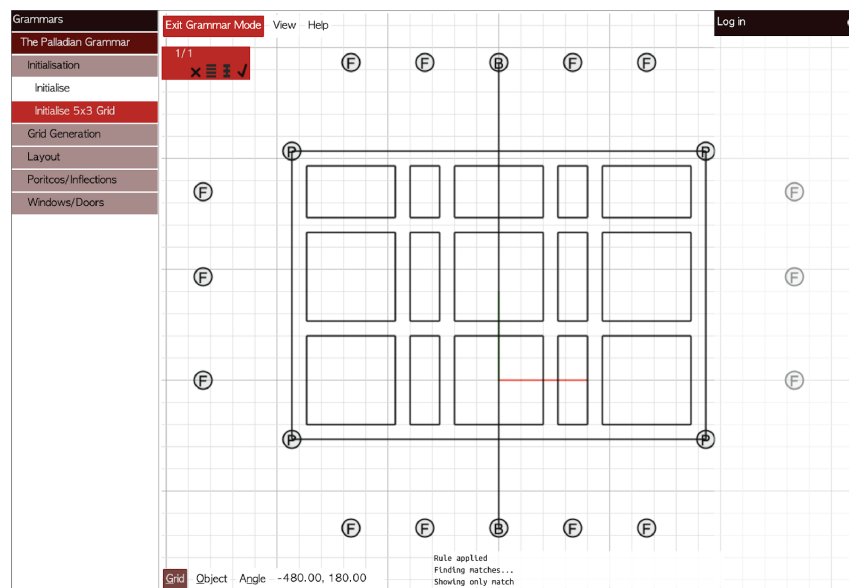


Figure 2. The Palladian Grammar webapp built by GRAPP, source: <http://grape.swap-zt.com/App>

Summary

The rule-based modeling method uses shape rules, selection rules, state rules, and algorithmic logic to generate layout configurations. It is used as the underlying generative method of several design grammars, these grammars were developed to explore the layout configuration of classical architectural styles.

To build a GD tool based on this method, developers need to define all the related shape rules, states, selection rules, and rule's implementation sequence (defined phases). The tool based on this method can only be implemented with a strict order in a grid.

In general, after this design method was introduced into the architectural design industry opened the golden period for the development of GD software in the architecture industry. Some famous GD research projects such as LOOS, SEED were born in this period.

2.2 Rule based modeling with search

To enable rule-based modeling GD tools automatically explore and test different layout configurations, CAD researchers try to add search algorithms and an evaluation system to the GD tool. With the help of them, these tools are capable of exploring layout configurations in a given space, and pick the most suitable generated layout in each iteration. One of the most famous GD software developed based on such a system is the LOOS (Flemming, Coyne, and Glavin 1987).

The LOOS

The LOOS is a rule-based modeling GD software which aims to assist architects to explore the furniture layout of small residential spaces such as kitchen or bathroom. Compared with a traditional rule-based design tool, the LOOS has two major improvements: it is no longer restricted by the grid, and it automatically explores all possible layout of furnitures in a given space through a search algorithm. The details as follows.

To eliminate the restriction from the grids, the LOOS team uses a new layout representation method called *loosely-packed arrangements of rectangles* that abstract shape rules and selection rules of a given design space (Coyne 1989). There are two parts of this representation method: first, it uses different sizes of rectangles to represent all geometries in a given space. For example a toilet in the bathroom will be represented as a rectangle bounding box. Second, it uses a 4 direction system to define all the spatial relationships between these rectangles. For example, for any pair of rectangles, there are four situations to define their spatial relations: above, below, to the left, and to the right. The developer created a set of symbols to represent these relationships such as use rule $a \uparrow b$ to represent rectangle a is above rectangle b.

To enumerate all the design alternatives with search, the team set up the system operation process like this: The system starts with a user adding a rectangle to the predefined coordinates in given design space. When two or more rectangles are added in the space, these two rectangles will form a spatial relation. If the user stops adding the rectangles, the search algorithm will try to find all possible spatial relations between these two rectangles and create intermediate design alternatives based on the current state. These alternatives will be evaluated by a spectrum design criterias and constraints defined by users to prune out the bad alternatives. The good results will be stored in a search tree as intermediate results for further exploration. ROOS asked the users to successively add rectangles to the system, meanwhile, the system will repeat the process of searching and evaluating so that in every state there is a best solution.

The LOOS team represents the set of relations that define a class of rectangles by a directed, arc-colored graph whose vertices represent rectangles and arcs represent spatial relations between selected pairs of rectangles. In general, this graph theory based model reflects all information for a single configuration.

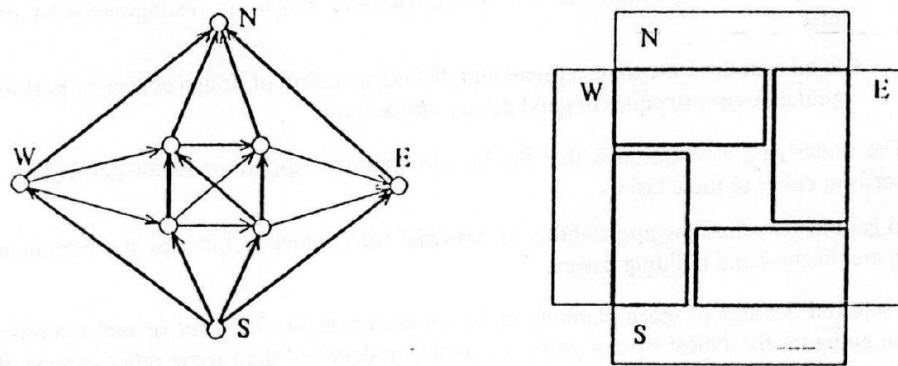


Figure 3. An orthogonal structure representing a loosely-packed arrangement of rectangles (Flemming, Coyne, and Glavin 1987).

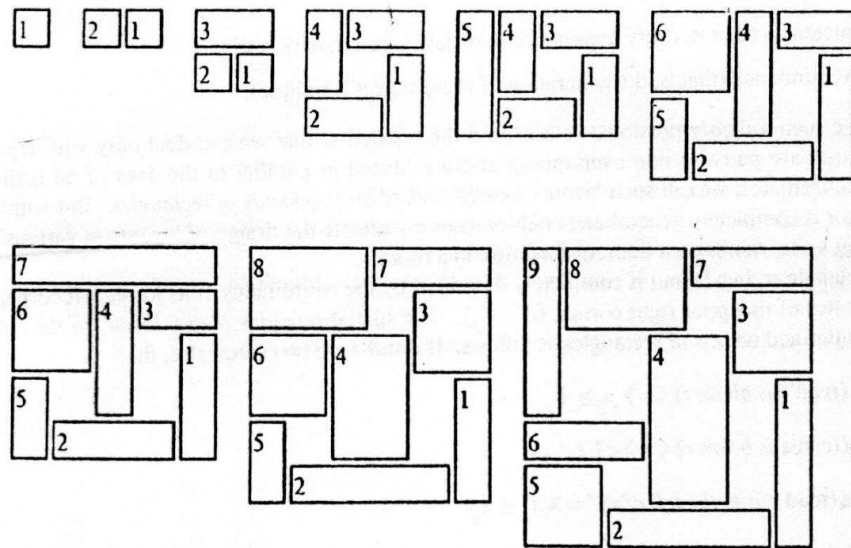


Figure 4. Layouts generated by successive application of generation rules (Flemming, Coyne, and Glavin 1987).

LOOS has successfully tested the idea of combining the rule-based modeling method with search algorithms, it greatly improves the scope of application of the GD software based on the rule-based modeling methods. After LOOS, a large number of GD software which targets small space layout design has been developed, some of these software are finally integrated into a large-scale GD software research project -- the SEED.

The SEED Projects

SEED is not a single GD software; it is a software environment which integrates multiple GD softwares to support architects in the early phases of recurring building design. In SEED, each of these GD software is called a module, it could be used to solve specific design problems. For example, the schematic layout design module could be used to generate recurring building floor plans. The generative method of all SEED modules is rule-based modeling with search algorithms.

The SEED team encourages the users to develop more modules for the system. To achieve this goal, they provide a GD software development process which they called *the use case-driven GD software development process* to guide all the developers.

Develop a rule-based GD software with use case-driven development process

The SEED team decided to design a process based on the Object-Oriented concept, because they think the concept of subdividing the big task into separate objects and encapsulating each of them would be compatible with the software architecture researchers tried on LOOS. After a thorough study on multiple object-oriented development methods the team finally selected the use case-driven development process as their choice for the entire SEED projects (F Robert 1993).

To understand this process we first need to understand the meaning of ‘use case-drive’. In general, a use case could be interpreted as a meaningful design task, it describes a sequence of actions or operations that must be executed together to achieve some design goal (Flemming, Erhan, and Özkaya 2004). For example, the task of designing a bathroom layout at a schematic phase is a use case. To achieve the design, a sequence of actions such as placing the toilets, placing the sink, calculating the ADA circle etc must be executed. I personally like to call this process ‘task-driven’.

According to the SEED project’s documentation, a typical use case-driven process is divided into five major steps:

- 1) **Initial specialization:** In this step developers should document basic system and context constraints.
- 2) **Use case development:** Developers need to produce a description of the desired functionality in terms of use cases in the end-users' language.
- 3) **Define static and dynamic Object models:** Static: developers need to produce class diagrams/UML models to understand the overall object models. Dynamic: developers need to produce sequence diagrams to understand user machine interaction.
- 4) **Coding:** produce code for all classes, use case-by-use case.
- 5) **Validation:** developers in charge of tests the code, use case-by-use case.

A image diagram depicts the entire process shows below

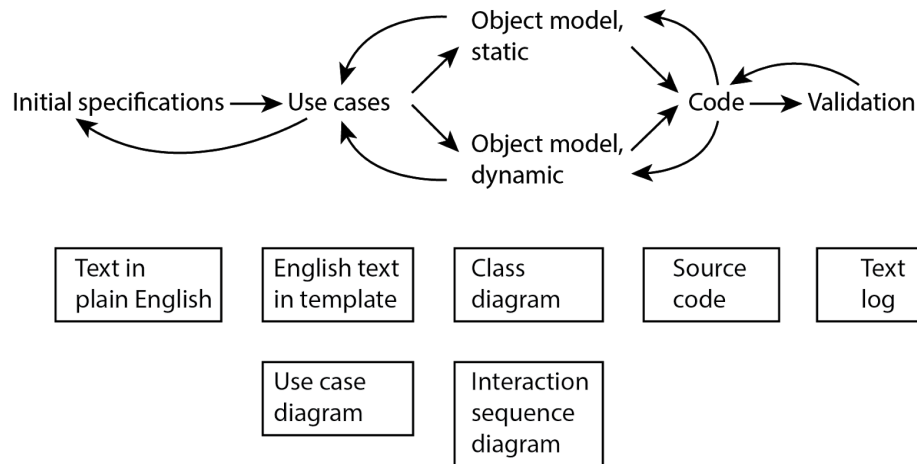


Figure 6. Steps and products of use case-driven software development (Flemming, Erhan, and Özkaya 2004).

The conceptual model of this process was initially completed in 1993 by a group of CMU researchers (F Robert 1993). It has passed the practical test of the entire SEED project and has been used as a standard CAD software development process in the AutoCAD plugin development course at CMU for a long time (Flemming, Erhan, and Özkaya 2004). Thus, this process is one of the few classic processes dedicated to guiding developers to develop GD software for architects. However, from today's perspective, we can find this process also has plenty of features that could be improved. A full analysis regarding the limitations of this process will be presented at the end of this chapter.

Summary of rule-based modeling with search

The rule-based modeling with search method uses shape rules, selection rules, state rules, and search algorithms to generate layout configurations for small interior layout configuration. The generating process is restricted to a defined order, the design space should be grid or semi grid.

There are some shortcomings of the method:

- 1) **It has to deal with a rectangle design space or grid system.**
- 2) **It is difficult to define the variables for the generative system.** To develop a rule-based GD software, the biggest challenge is to accurately decompose a space layout into a set of shape rules or modules that can represent all the information of the given layout. It requires a high level understanding of the related architectural design knowledge.
- 3) **It doesn't optimize the solutions.**
- 4) **It's slow.** Ideally a GD software based on rule-based modeling method is capable of enumerating all the possible layout configurations because the search algorithm is required to search all spatial relations step-by-step. However, based on this mechanism search algorithms will make a lot of meaningless attempts and ultimately lead to software inefficiency. This problem has not been solved for a long time.

These limitations motivate the CAD researchers to explore different types of generative methods to solve the layout problems. Among these methods, the most popular one is called variational modeling with optimization.

2.3 Variational modeling with optimization

Based on parametric design and evolutionary solver, the variational modeling with optimization (hereinafter called the VO) method uses an explicit function to translate a set of predefined parameters to a design solution. Each variation of parameters can generate a solution, the parameters should be numerical. The optimization function of the method is provided by an evolutionary solver. It is an implementation of a special type of metaheuristic search algorithm that uses concepts related to “biological evolution” to discover possible solutions to an optimization problem.

There are three major differences between VO and rule-based search: first, **the variables** for generating solutions. A VO based GD tool produces the variation of a design solution through varying the value of numerical parameters while a rule-based GD tool produces the variation through change rules or shapes/modules. Second, **a numerical design goal**. VO based GD software can only apply to the design problems with numerical design goals. Third, **capability of optimization**. VO based GD tools are able to optimize the generated results based on its evolutionary solver. An evolutionary solver usually consists of either Genetic or simulated annealing algorithms.

Genetic algorithms

The Genetic algorithms (hereinafter called the GA) are one of the commonly used algorithms of evolutionary solver, it's based on the mechanics of natural selection and natural genetics (Goldberg, 1989). A typical GA generally encodes the information about the problem in the form of a string of values. This string that holds the information is called a “chromosome”. A chromosome is made up of several smaller pieces of data called “genes”. When a GA is initialized, a number of chromosomes are randomly generated. This collection of chromosomes is termed as a “population” (RAO and OZEL 2004).

The process of natural selection starts with the selection of fittest chromosomes from a population. The way it selects the fittest chromosome is through a combination process including

scoring each chromosome after initialization, then uses three generic operators: selection, mutation, crossover to select and further recombine the chromosomes with the highest score. The chromosome produced in this selection process inherits the characteristics of the parents and will be added to the next generation. This process keeps on iterating and at the end, a generation with the fittest individuals will be found.

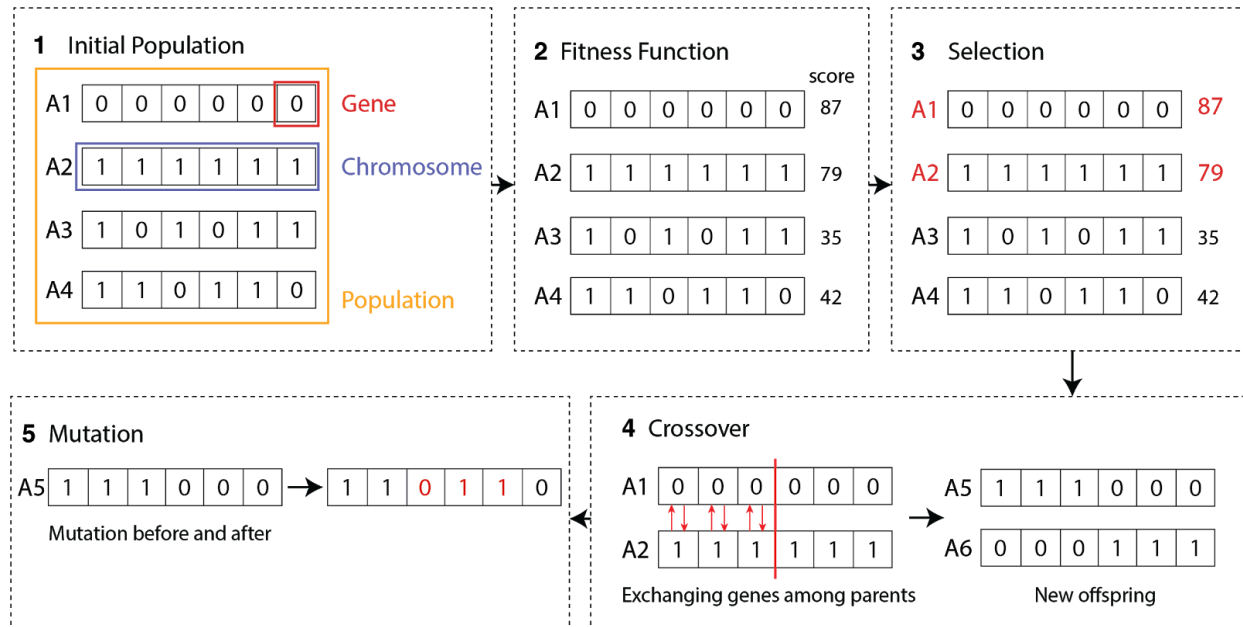


Figure 7. Typical five phases of a genetic algorithm,

source: <https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code>

This notion of natural selection and generation can be applied in the field of architectural design. For example, we can treat each chromosome as a design solution, it encodes the design rules and the parameter values as its Gene. The encoding is in binary form, and is unique to the design problem. To build a VO based GD software, the most challenging step is to find such a Gene that can significantly influence the design results.

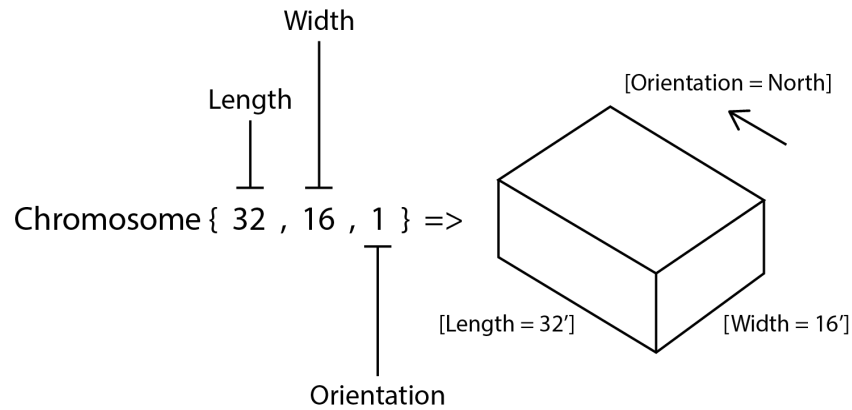


Figure 8. Translating the binary values from the chromosome into drawing forms (RAO and OZEL 2004).

VO based GD software

We usually use Galapagos or Project Refinery as an VO based example. However, these two are actually evolutionary solvers rather than a complete GD software that is capable of producing design results by themselves. The most popular way to develop a VO based GD software is to use one of these tools as an evolutionary solver, and then build the parametric modeling part based on a visual programming platform. A good example that is developed in this way is the office layout generator developed by Autodesk (Smith, Williams, and Thorley 2019).

The goal of this prototype is to find a solution that maximizes the number of desks in the given office layout. The prototype is developed on a dynamo platform, with project refinery as its evolutionary solver. To generate the layout, the prototype asks the users to input five defined static inputs including: floor plate, a perimeter curve that represents the boundary of the office, the desk's width and depth, and back-to-back tolerance of the desk. To maximize the desk number, the prototype uses both the boundary walls and room walls as the variable inputs.

Once the project refinery starts to search for a solution, the room wall will move along the boundary wall with a tolerance distance. This movement causes the neighborhood sizes to change, providing new layouts respectively.

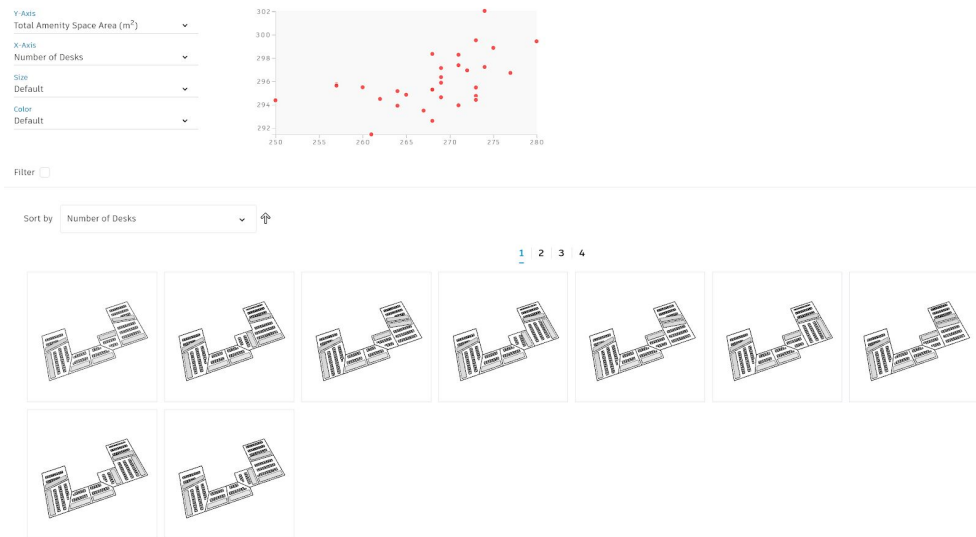


Figure 9. The design output of the office layout generator. Source: <https://www.generativedesign.org/>

Summary

The variational modeling with optimization method uses an explicit function to translate a set of predefined parameters to a design solution and an evolutionary solver to optimize the solution. Currently the GD software built by this method is mainly used to generate medium-sized layout such as office layout.

To build a GD tool based on this method, developers need to define all the related static parameters, variable parameters, and a clear quantifiable design goal. The most critical steps of building such a tool is to accurately define the design goals, static and variable parameters.

The one major shortcomings of the method is the stability of the GA. Sometimes genetic algorithms don't have a clear convergence direction.

2.4 Simulation-based system

Simulation-based system could be divided into two parts: discrete simulation, and continuous simulation. The one I'll focus on in this thesis is the discrete simulation because this part is more widely used in the architectural design industry. A typical simulation-based generative method usually uses a computational method called cellular automata to model and simulate the building layout generation in a grid or graph (Velooso and Krishnamurti 2019).

Compared with the rule-based method, the Simulation-based system is much harder to control during the process of layout generating since it updates the entire layout at each timestep. It is a relatively high performance generative method which could be applied in large scale architectural layout design problems such as residential building site planning, and urban planning.

Cellular Automata

As a computational method, cellular automata can be incorporated in both modeling and simulation contexts (Khalili Araghi and Stouffs 2015). A typical CA consists of grids, each cell of the grid is one of a finite number of states. The process of growth is based on the state of individual cells, which is determined by its neighbor's state at a previous time and related local rules. At each timestep, the system is updated.

Conway's game of life can be a good example to explain a simulation-based generative system. Introduced by John Horton Conway in 1970, Each cell is in one of two possible states, alive or dead. Every cell interacts with its eight neighbours following these rules:

- 1) Live cell with < 2 live neighbours dies, as if caused by underpopulation.
- 2) Live cell with 2 or 3 live neighbours lives
- 3) Live cell with > 3 live neighbours dies, as if by overpopulation.
- 4) Dead cell with 3 live neighbours becomes a live cell, as if by reproduction.

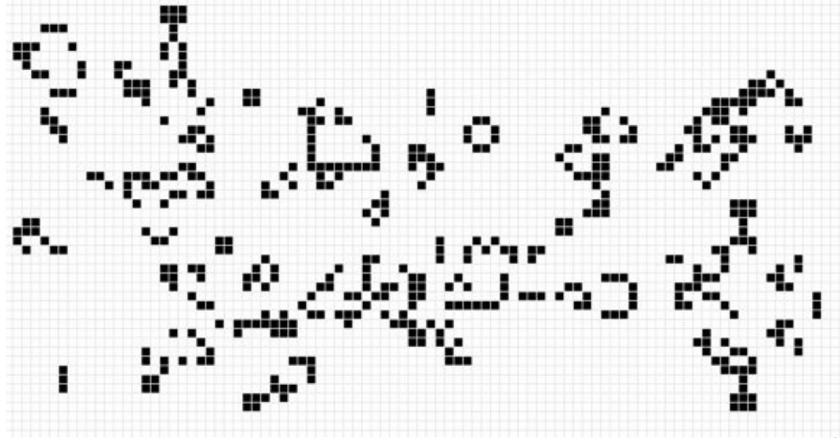


Figure 10. An example rendering of Conway's game of life.

Simulation-based system in architecture

I picked the prototype developed Salmon and Rudi's as an example to show how the Simulation-based system could be applied in architectural layout design problems. This prototype uses cellular automata (hereinafter called the CA) as its underlying generative method to generate a variety of building forms for high density housing at the early stages of the design process. It mainly focuses on each house's accessibility and lighting requirements. The typical modeling and generation process is as such: step 1, user define the footprint and grid; step 2, user determines circulations; step 3, system assigns random initial configuration on each level; step 4, Generator will start the simulation process (Khalili Araghi and Stouffs 2015). The simulation process divided into two part:

1. **Investigation.** System investigate the type and state of each cell in terms of accessibility, natural light and density
2. **Cell updates.** Cells remain alive if it has access to proper circulation, global light and proper neighborhood density, Else dies.

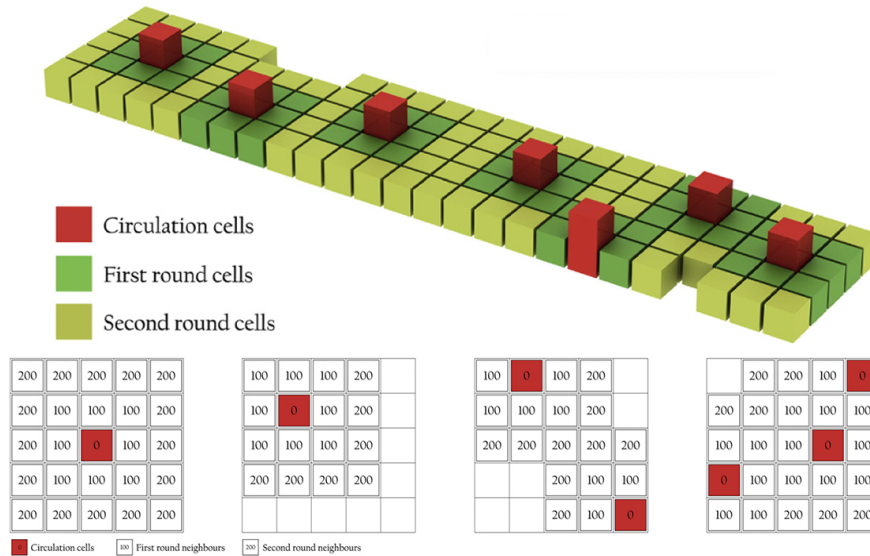


Figure 11. Above, 3D illustration of neighborhood patterns in 5 x 25 grids of cells. Below shows the simulation procedure once its start (Khalili Araghi and Stouffs 2015).

Summary

The simulation-based modeling method uses a cellular automata to model and simulate the building layout generation in a grid. The layout will be updated in each timestep. GD software built by this method is mainly used to generate large scale layout design such as urban plan or highrise plan.

The requirement for building a GD tool based on this method is similar to rule-based modeling except that this system requires a cell interaction rule between one cell with its neighbors.

The one major shortcomings of the method is that the generated solutions from this method usually need much more manipulation compared with other two methods because its generative procedure is difficult to control.

2.5 Reflection on the Case Studies

After studying and analyzing the characteristics of the 3 major generative methods with their related GD prototypes, I have compiled the information regarding application scenarios, variable inputs, and implementation requirements for each method in the chart below. I called it Method Selection Table. With this table, when we develop GD software for a given task, we can choose an appropriate method according to the characteristics of the task itself.

Method Selection Table

	Application Scenarios	Variable inputs	Requirement
Rule based modeling + Search Palladian Grammar, LOOS, SEED	Small layout tasks: Kitchen, Bathroom, Building Core. The task has to be generated in order. Must have a solution	Shapes or modules. for growth State of modules. Rules for growth	Quantifiable design goals Grid or semi-grid
Variational modeling + Optimization Galapagos, Refinery	Small to medium layout tasks: Office floor plans. Parking layout plan. The task need optimization	numerical parameters as Genes	Quantifiable design goals An explicit function translate parameters to solution
Simulation-based system High density Housing planner	Medium to large layout tasks: High density house neighborhood. Urban plans	cells for growth State of cells.	Grid Local rules determine the cell transition

After reviewing all the case studies, I went back to the *use case-driven development process* and started thinking about what is missing from the process and how we should improve it so that it can adapt to today's industry environment. Below is what I summarized the missing contents of the use case-driven process:

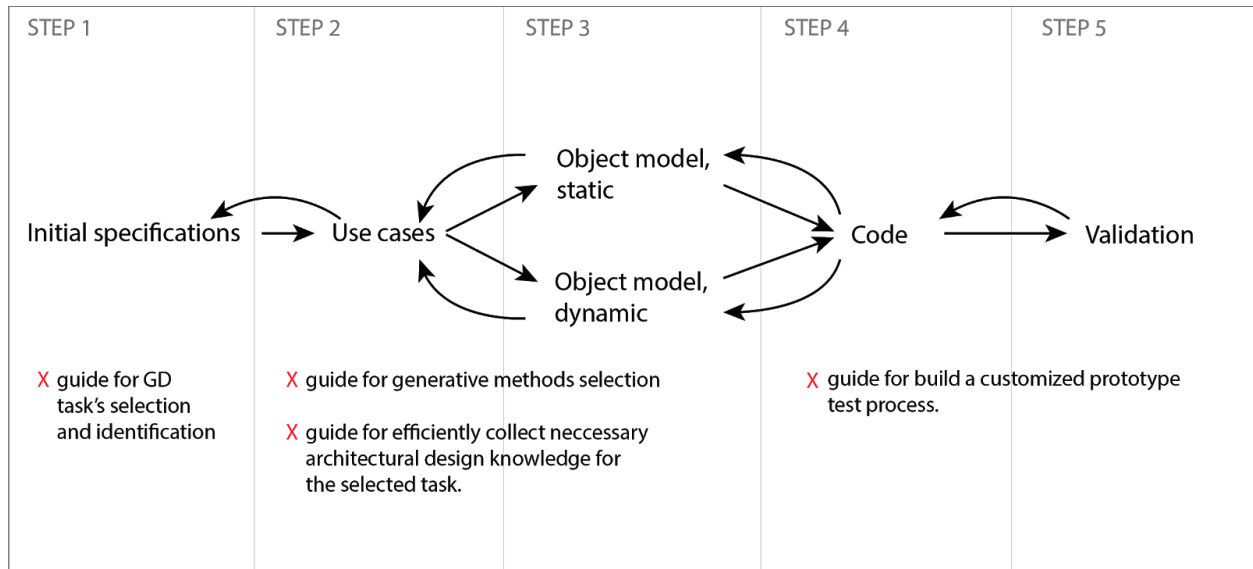


Figure 11. The contents that is missing from the use case-driven development process

- 1) **In step 1, it doesn't provide information regarding how the use case is being selected for development.** The current function of step one is only about setting up task constraints and system environment. However, as we learned from the case studies, to pick an appropriate design task such as a task with quantifiable design goals to develop GD software is very important.
- 2) **In step 2 & 3, the step of choosing the appropriate generative methods is missing. In addition, it doesn't provide enough guidance regarding how to efficiently acquire related architectural design knowledge for the selected task.** The current process is purely based on the rule-based modeling which greatly limits the scope of application of this process. In addition, as a development guide for developers of various backgrounds, it is important to provide a method of how to quickly obtain necessary architectural design knowledge for the selected task.

3) In step 4 & 5, it doesn't provide guidance regarding how to test the performance of a GD software. Evaluating the value of GD software mainly depends on whether the design results it generates meet the needs of architects. A GD software development process needs to provide developers with a process and method to test the results generated by the GD software.

Based on the above findings, we have a preliminary idea on how to design a new GD development process. In the next chapter, I will introduce the details regarding how the new process complements the missing information of the use case-driven process.

Chapter 3. Hypothesis: A GD software development process

In this chapter, I proposed a new GD software development process. This new process consisted of four major steps: Step one, Identify the appropriate design task to apply GD technology. Step two, ideate the most suitable generative method for the selected design task. Step three, develop the prototype with a visual programming platform. Step four, test prototype's performance with customized dimensions. I'll give an explanation for each step in detail.

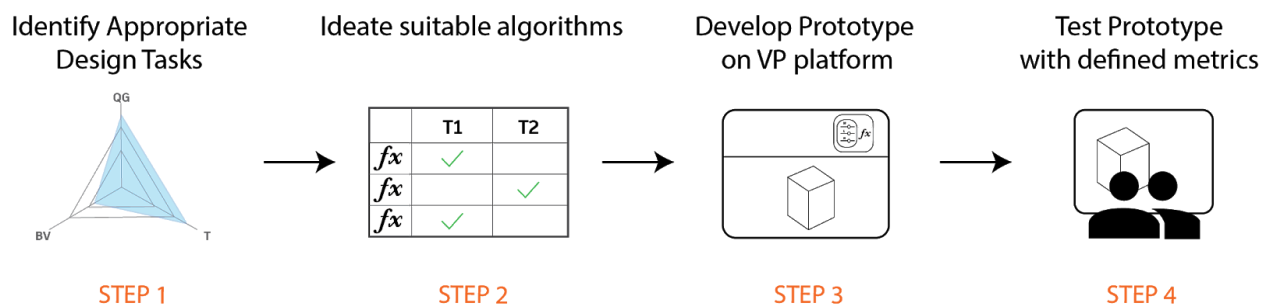


Figure 12. Steps of the proposed GD software development process

The diagram below shows the changes in the structure and content of each step of the new process relative to the use case-driven process: **Step 1.** add task selection guidance on the original. **Step 2.** integrates the original step 2 and 3, add the guidance of methods selection and acquiring architectural knowledge. **Step 3.** integrates the original step 4 and 5, replaced the code template. **Step 4.** A new added step for prototype testing guidance.

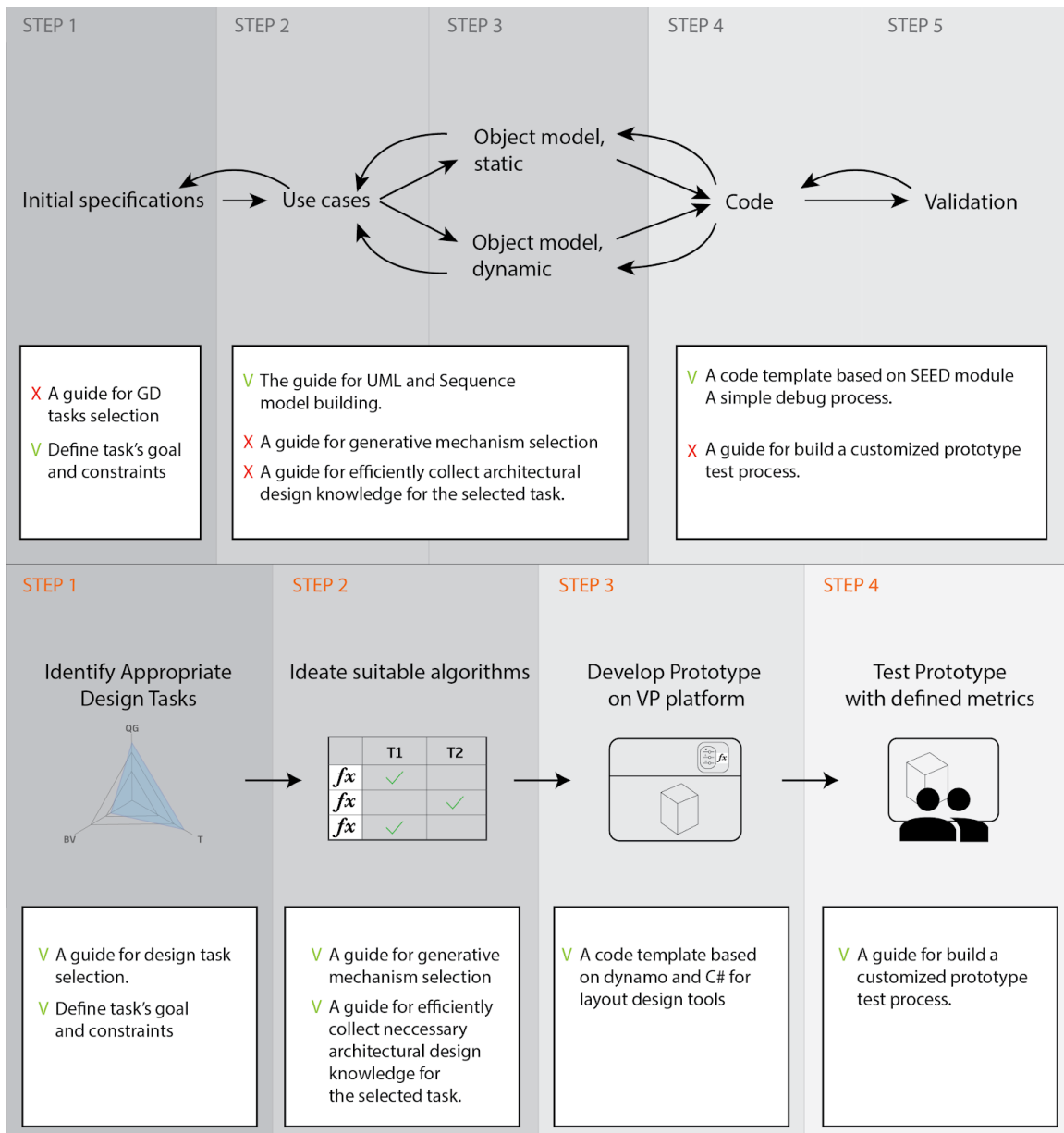


Figure 13. Steps and products of use case-driven software development

3.1 Identify appropriate design task to develop GD software

An architectural design task that is suitable for developing GD software has the following three characteristics : 1).The task has clear, quantifiable design goals. 2).The current design method of using traditional CAD software to complete the task is tedious and time-consuming. 3).The task itself has high commercial value.

The way to find a suitable task divided into two steps: find a task category, then find the most suitable sub-task of the category to develop GD software. For example, a basement design is a task category, it includes sub-tasks like fire compartment layout, MEP and structural design, parking layout design etc.

Find a task category

- 1) **Conduct an online survey.** In this survey all the major architectural design task categories should be listed out and each participant is limited to vote for 3 categories. An extra text area at the bottom of the survey will be served to allow participants add the task that is not on the list. We will dynamically update the task categories based on the results we collected. After the results collection, we can rank the results and pick top 3 for the user interview.
- 2) **Conduct user interviews.** We will arrange user interviews in either group sessions or one to one in order to find out whether the category has high commercial value and if there are competitor products in the market. We will pick one category out of three after this process.

Narrow down to a specific subtask

The quickest way to find out all the subtasks of the category is to draw a user-journey-map. A user-journey map will indicate all the potential subtasks of the category. Take each subtask from the map and measure it with the three characteristics we defined above. Select the subtask that meets all the characteristics we defined above (The summary of interview questions and findings is documented at Appendix).

3.2 Ideate the most suitable generative method for selected task

There are three sub-steps in this section:

- 1) Make a preliminary method selection based on the **Method Selection Table** we defined in chapter 2.
- 2) Study the selected task, define static and variable inputs, evaluator, design space, simulation or growth rules if needed, and related functions if needed. If we can not define either variable inputs or growth/simulation rules for the selected design task, go back to step one and change to another method.
- 3) Make a final decision on generative method choice.

If we choose rule-based modeling or simulation-based systems, we need to accurately define all the shapes and related shape rules which are needed to generate the task layout. If we choose variational modeling with optimization, we need to find out the key variables that can be used as ‘Genes’ of genetic algorithms.

To accurately define the variable inputs requires a deep understanding of the selected design task. A good way to acquire the architectural knowledge for the task is to talk with the design department of real estate companies. Sometimes this department has its own research group that purely studies specific architectural design tasks. For example, Poly Real Estate has their own basement design research group.

To quickly define all static inputs and design criterias for the chosen task, a good way is to collect these two materials: the architectural design index chart, which is provided by architects, and the design review table, which is provided by clients. These two materials almost covered all the core information regarding a design project.

3.3 Develop prototype with visual programming platform

A good place to start prototyping is a visual programming platform (hereinafter called the VVP) because it provides plenty of pre-wrote modules that handle different geometric operations for developers to refer to, and it comes with sophisticated evolutionary solvers for developers to use it as the optimizer of the prototype. In addition, a VVP is connected with mainstream modeling software, which makes the data exchange between prototype and modeling software more stable.

There are two major visual programming platforms in architecture: Dynamo and Grasshopper. As an experienced user of both ends, I personally recommend developers go with Dynamo with reasons as follows:

- 1) The latest version of Dynamo has been separated from revit, which makes the running speed of dynamo significantly improved.
- 2) Dynamo is an open source tool. Developers could easily access plenty of useful code templates.
- 3) Dynamo is fully connected with visual studio. Developers are allowed to code in a professional IDE and visualize the result in dynamo.

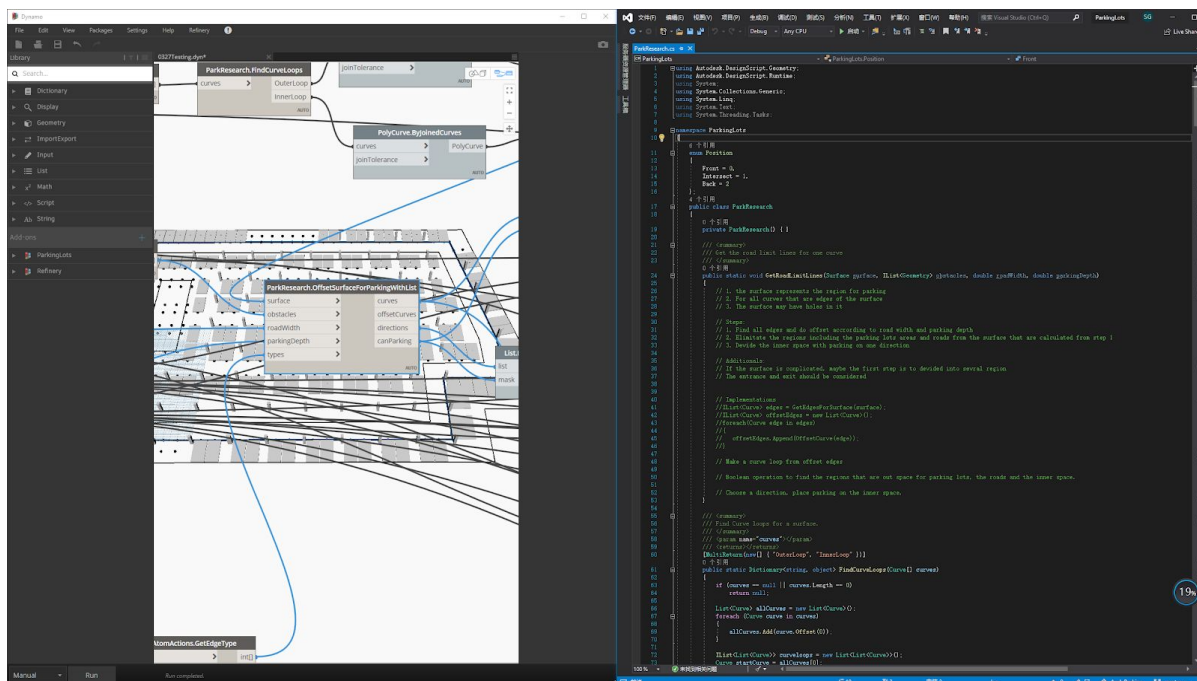


Fig 14. Development with dynamo and Microsoft Visual Studio.

3.4 Test prototype's performance with customized dimensions

When setting evaluation criteria for the prototype, we first consider the role of the GD software. Currently there are two recognized functions of a GD software: using algorithms to assist architects in design solution exploration, and improving the modeling efficiency on some tasks that are not handled well by traditional CAD tools (Davis Davis 2020). Based on its role, we can evaluate a GD software with two dimensions:

- 1) **The ability of the software to restore a design solution.** We need to see how big the gap is between the solution generated by the GD software and by architects. We should set the error rate according to different tasks. When the generated solution is within tolerance, we consider the prototype to be valuable.
- 2) **The ability of the software to explore new solutions.** We need to look at the pass rate of the generated results for a given task. Meanwhile, we should observe whether the GD software can provide better solutions than architects in the same amount of time for solution exploration.

To test a prototype, developers can either build several models to test it by himself, or he can do a demo session with groups of architects so that architects can participate in the testing.

Chapter 4. Methods

In this chapter, I developed a GD prototype for a basement parking layout design task in order to demonstrate the feasibility of the process. Based on the generative method of *variational modeling and optimization*, this GD software can generate a variety of parking stall's layout in a very limited amount of time and is capable of maximizing the total number of stalls for each design alternative. After testing the prototypes with two dimensions including the ability of the software to restore a design solution and the ability of the software to explore new solutions, the prototype is proved to be valuable. I documented the entire development process and the related dynamo scripts for future reference.

4.1 Select basement parking layout design task

I started with creating a questionnaire to find out which design task architects most want to have GD tools. I received replies from 58 architects in 5 cities. They picked 15 tasks on the list, among which the top three are panelization studies, basement parking layout design, and program layout design.

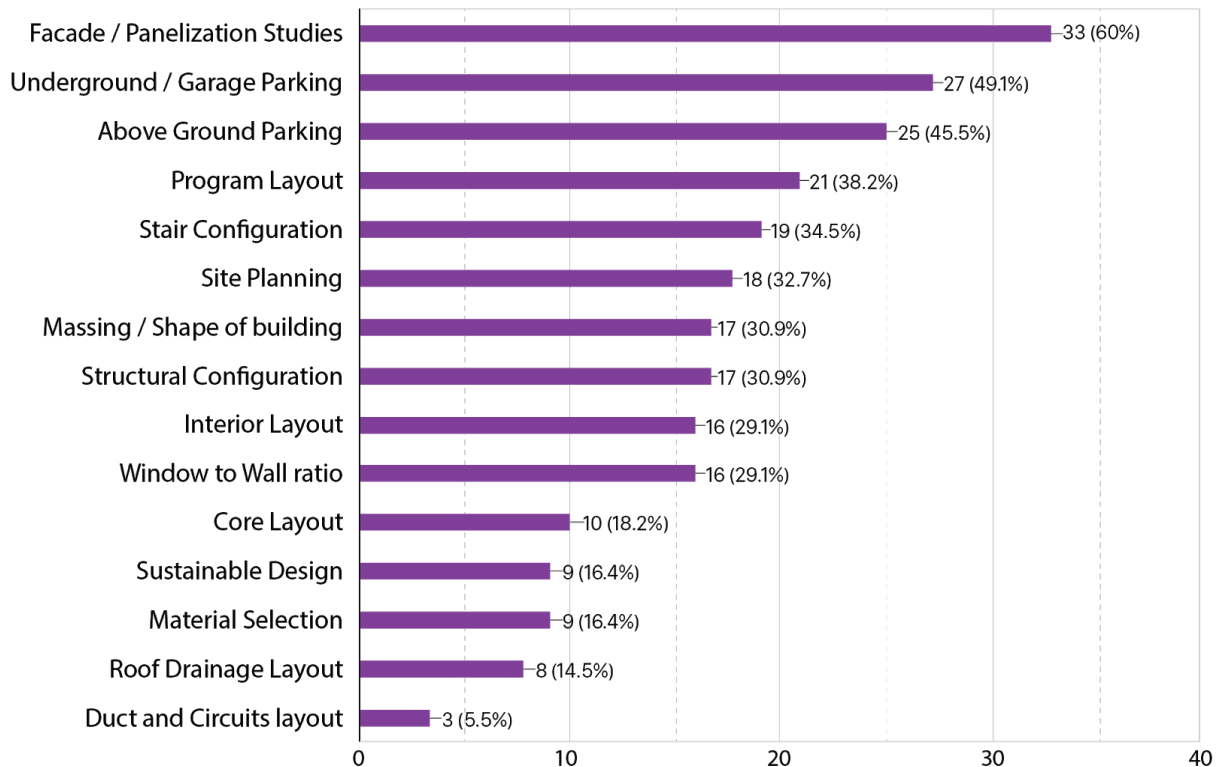


Fig 15. Online questionnaire for architects to select the top three categories which they want a GD tool. n = 58.

I evaluated the three tasks in terms of commercial value and whether there are potential competitors in the market and found out that the basement parking layout design is the most suitable task to develop a GD tool. The reason is that this task has not been fully researched by leading CAD software companies, and it does have relatively high commercial value compared with other two tasks according to Poly Real Estate (Xiong 2017).

Then I hosted 5 interview sessions with groups of experienced architects in order to understand more details about the basement layout design task. I found out that this task usually consists of three sub-tasks: 1. Divide the basement into different functional areas such as MEP

rooms, motor vehicle parking areas, bicycle areas, and core. 2. Place parking stalls in the motor vehicle area. 3. Add MEP and fire-defense facilities into the basement. Among the three sub-tasks, the one has the highest priority to develop a GD software is placing the parking stalls in the motor vehicle area, reasons as follows:

1. The goal of basement layout design is to maximize the parking stall's number in a given space.
2. Architects have always complained that placing parking stalls is a tedious, and time consuming task. Because each layout iteration will be accompanied by the rearrangement of parking stalls, and arrangement of parking stalls requires a lot of manual copy and paste.

Thus, I decided to develop a GD software that is focusing on generating the basement parking stall's layout for architects.

4.2 Select generative method for basement parking layout design task

Now we have to choose the appropriate generative method for the basement parking layout design task. This task has the following three characteristics: it is a small to medium-sized layout design task, it comes with a quantifiable design goal, and it requires optimization. Based on the Method Selection Table, my initial choice is variational modeling with optimization. Then I started to do an in-depth research on the task in order to define the static and variable inputs for the generative method.

	Suitable for Medium size	Provide Function of optimization	Has essential Variable inputs
Rule based modeling + Search	YES	NO	?
Variational modeling + Optimization	YES	YES	?
Simulation-based system	NO	NO	?

Define static inputs

To define static inputs, I studied a basement project index-chart from a leading architectural design company and summarized all the static inputs below :

- 1) The basement boundary : we need a boundary curve to define the entire design space.
- 2) The position and size of all the columns : we need a file containing information regarding columns sizes and locations.
- 3) The position and shape of all non-parking spaces (MEP rooms, ramps, etc) : we need a file containing information regarding geometries of non-parking spaces.
- 4) The length and width of the parking stall : the default value is 3000 x 6000mm.
- 5) The width of both main road and auxiliary road : the default value is 7000mm.

	A	B	C	D	E	F	G	H	I	J	K
1	Basement Index Chart										
2	Total Area(A) (m ²)	28069	Basement Parking Requirement(D)	1113	Fire compartment Requirement (m ²)	10100	Level height		3.5		
3	Complimentary basement	0	Site Parking number(E)	353	Actual fire compartment area (m ²)	10100					
4	Other Area(C) (m ²)		Basement Parking Number(F)	760	Local Stall Size		Main Grids (mxm)		7800x8100		Others
5	Basement One Stall Index(A-B-C)/I (m ²)		Project One Stall Index (A-B-C)/ (C+I) (m ²)	25.22	Actual fire compartment area / Total Area	35.98%					24.00%

Fig 16. A typical Basement index chart.

Define variable inputs

To define variable inputs, I've studied several Real Estate company's basement design guidebooks. Based on the study of a module evaluation system and a boundary-driven design strategy provided by Poly Real Estate (Xiong 2017), I finally defined two variable inputs: **the number of boundary walls which can be used for parking**, and **the orientation of internal parking stalls**.

Poly believes that all basement parking stall's layout configurations can be represented by 7 different rectangular modules. These modules are divided into two groups: the wall-mounted group and the road-mounted group. There are five modules in the wall-mounted group, such as the corner-parking module, the sidewall parallel parking module, and the sidewall

perpendicular parking module. The road-mounted group has two modules: the mainroad perpendicular parking module, and the auxiliary road perpendicular parking module.

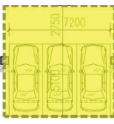
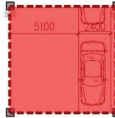
Module 1 	Vertical back to back parking Area : 61.23 m ² Efficiency : 20.41 m ² /Car	Module 2 	Vertical back to wall parking Area : 62.79 m ² Efficiency : 20.93 m ² /Car
Module 3 	Vertical back to wall bilateral parking Area : 87.13 m ² Efficiency : 29.04 m ² /Car	Module 4 	Corner Parking Area : 132.92 m ² Efficiency : 33.23 m ² /Car
Module 5 	Vertical back to bilateral parking Area : 94.00 m ² Efficiency : 31.33 m ² /Car	Module 6 	Parallel Wall Parking Area : 47.04 m ² Efficiency : 20.41 m ² /Car
		Module 7 	Vertical back to bilateral parking Area : 40.17 m ² Efficiency : 40.17 m ² /Car

Fig 17. Poly defined these seven modules to represent any parking layout configuration in a common basement (Xiong Daohui, 2017).

Based on their module system, poly recommends a strategy to use these modules to create basement layout configuration. the strategy consists of two steps:

- 1) Place the modules from the wall-mounted group to the boundary of the design space. Architects can choose any side of the boundary to place the module.

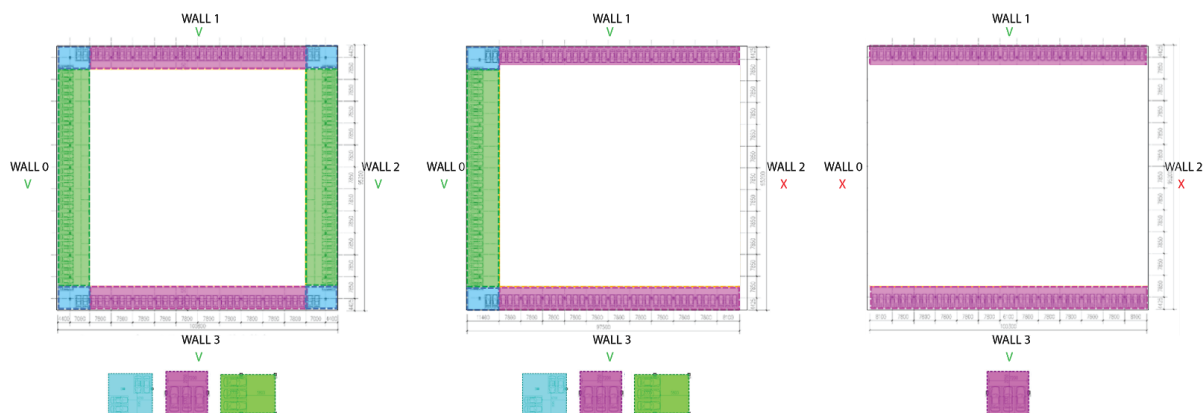


Fig 18. Different ways to handle boundary parking.

- 2) Place the modules from the road-mounted group to the center of the design space.
Architects can decide the orientation of these modules.

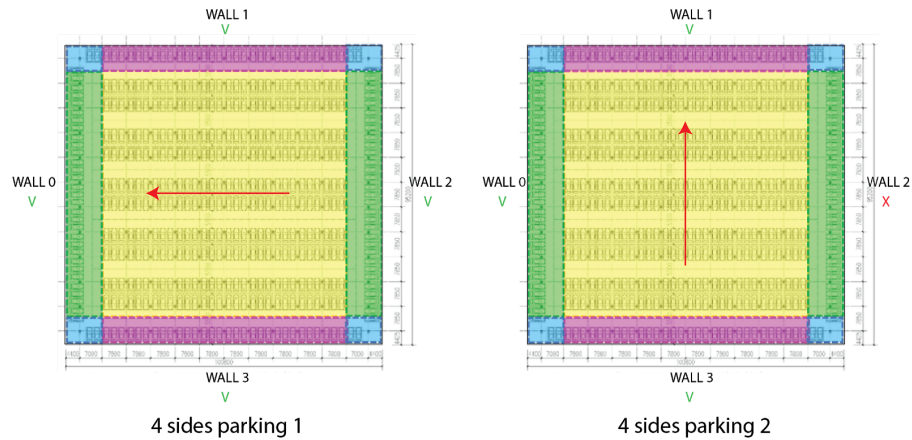


Fig 19. Different ways to handle internal parking.

From Poly's strategy we can tell that the parking stalls number could be determined by two factors: the number of boundary walls with parking stalls next to, and the orientation of the internal parking stalls. These two factors are defined based on the years of experience in basement design by the Poly research team, we think it is reasonable to use it as the variable inputs for the prototype.

To make these variable inputs to be numerical so that they can be represented as Genes in the genetic algorithms, I used a binary string to represent the parking conditions of all boundary walls.

The walls with stalls around it will be represented as 1, otherwise 0. A string 1111 represents a basement with parking stalls on all four sides. A string 0101 represents a basement with parking stalls around wall 1 and wall 3. We can imagine that in a four-side basement, there are a total of 15 kinds of boundary layouts. Meanwhile, each boundary layout corresponds to two internal layouts. The orientation of the internal layout was also represented as a binary number. These variable inputs will form the chromosome for the genetic algorithms.

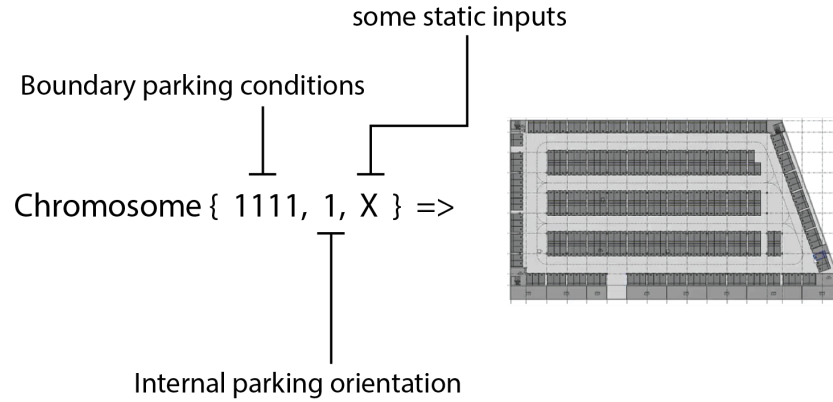


Fig 20. The chromosome prepared for the genetic algorithm.

Design result evaluator

After we define both static and variable inputs, the last information we need to collect is the design criteria for result evaluation. As the process suggested, we will collect these information from the design review table. However, in our case, the tool we are going to create only handles the stall's layout, thus the only feature we are going to measure is the stall's number.

Basement Design Review Table									
1	Basement Size	Whether the size of basement area is strictly controlled, the number of allocated parking Spaces only just meet the requirements of application for approval and practical functions	Agree ☐	Modify ☐	9	Entrance/Exit	Whether the ramp (including unidirectional and bidirectional) are perpendicular to the layout of the driveway.	Agree ☐	Modify ☐
2		Whether the site parking has been fully Utilized first reduce the scale of basement.	Agree ☐	Modify ☐	10		Whether other interstorey ramps have been set up according to the minimum principle of current specifications except the entrance ramp out of the ground floor.	Agree ☐	Modify ☐
3		Whether the contour of the basement has made full use of the shrinking condition, whether the number of basement floors has been fully optimized	Agree ☐	Modify ☐	11		Whether the net width of single-vehicle ramp is 24.0m and that of dual-vehicle ramp is 27.0m for the entrance ramp out of the ground.	Agree ☐	Modify ☐
4	Basement Boundary	Whether the basement contour has been removed unnecessary folding and sharp Angle to make the contour as square as possible	Agree ☐	Modify ☐	12	Fire Compartment	Whether the basement protection area is centrally designed against the side wall and placed at the lowest level.	Agree ☐	Modify ☐
5		Whether invalid space has been eliminated.	Agree ☐	Modify ☐	13		Whether the division of the air defense wall in the basement satisfies the principle of the shortest length and the least number of air defense doors and sealing plates.	Agree ☐	Modify ☐
6		When the contour line of the local library passes by the vertical members of the original tower, is it possible to coincide with the vertical members and use the vertical members as the side wall of the basement?	Agree ☐	Modify ☐	14		Sales Optimize	For some buildings without basement, if underground passage is setup to connect with basement to improve traffic convenience	Agree ☐
7	Module	Whether the bilateral parking way of the lane has been fully used to arrange parking Spaces, to avoid unilateral parking (including vertical and parallel parking) of the lane, and to strictly control the low-efficiency area of pure lane without parking space.	Agree ☐	Modify ☐	15	If the grid size meet the requirement		Agree ☐	Modify ☐
					16	Whether the single stall efficiency meet the requirement	Agree ☐	Modify ☐	
					17	Whether the height meet the requirement	Agree ☐	Modify ☐	
					18	Other Requirement		Agree ☐	Modify ☐
					19			Agree ☐	Modify ☐
8		Whether to arrange parking Spaces along the long direction of the basement as far as possible on the whole, and avoid arranging parking Spaces along the short direction.	Agree ☐	Modify ☐	20			Agree ☐	Modify ☐

Fig 21. A typical basement design review table

After we collected all the information regarding static and variable inputs, evaluation criteria, we should go back to check if the original choice of generative method is still good. Apparently, based on the table below, the variational modeling with optimization is still the most ideal generative method for this basement parking layout design task since it satisfies all the dimensions. We will keep staying on the choice.

	Suitable for Medium size	Provide Function of optimization	Has essential Variable inputs
Rule based modeling + Search	YES	NO	YES
Variational modeling + Optimization	YES	YES	YES
Simulation-based system	NO	NO	NO

4.3 Develop prototype based on variational modeling with optimization

The process of building up this prototype divided into five steps: 1). set up generative modeling logic; 2). build up a UML model; 3). set up the static and variable inputs as a dynamo format slider; 4). build up the parametric modeling scripts for generating the entire layout; 5). make connection between scripts and project refinery, which is in charge of result's optimization.

Step 1: Set up generative modeling logic

The generative logic consists of two parts: find the parking curves based on boundaries, then find the parking points based on parking curves.

Find the parking curves based on boundaries

- 1) Select boundary curves that architects want to arrange stalls next to. Make sure the curve length is 3 times longer than the stall's width. Offset these curves to the internal space with a distance of the stall's length. Store these curves for boundary parking.
- 2) Make a new boundary based on the existing boundary curves which have not been selected and the offset curves. let's call this new boundary A
- 3) Offset the boundary A to the internal space with a distance of the main road's width. Now we get a new boundary B.
- 4) Select the longest boundary curve from boundary B, make an array of this curve to the opposite side, store these curves for internal parking.

Find the parking points based on parking curves.

- 1) Split the curve into multiple sub-curve if the curve interacts with any obstacles geometry.
- 2) Split the sub-curve into stall-width curves, find the central point of these curves. Make sure to keep the orientation information of all points.
- 3) Place the stalls on these points.

Some customized tools to handle multiple-direction curve's extension, curve's interaction, multiple-direction offset is always needed. I coded all these necessary tools and included them in the code template I shared on the github.

Step 2: Build up UML model

The UML model helps developers understand the overall structure of the software. Based on this model, we can easily distinguish the objects or functions we need to code on our own and the functions we can acquire from dynamo libraries.

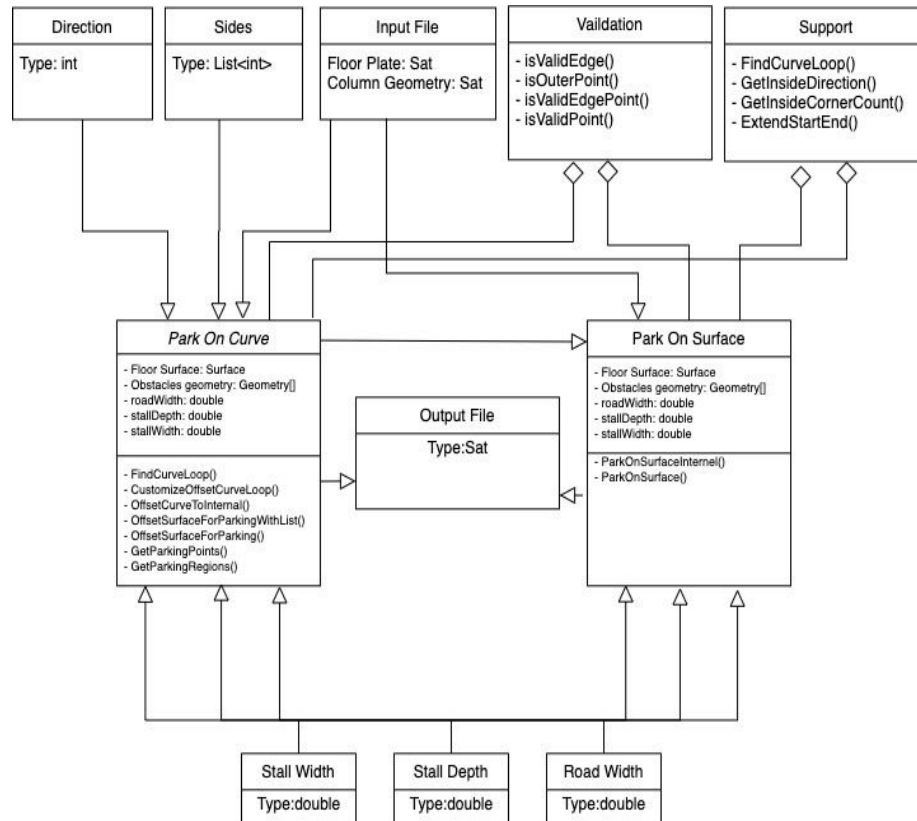


Fig 22. The UML model for the variational modeling based GD prototype .

Step 3: Set up static and variable inputs as dynamo format slider

Here is the description of what types of files and units you need to input to each of these slides

Static inputs

- 1) The floor plate : sat file contains floor plate from revit, hole is allowed
- 2) The column geometry : sat file contains all the columns
- 3) Road width : mm, default value from index chart
- 4) Stall width and length : mm, default value from index chart
- 5) The obstacle area : sat file contains non-parking area

Variable inputs

- 1) Type : User can input one decimal number in this slider, it will be translated into a binary string which represents the boundary parking conditions.
- 2) Edge no parking : user can enter the serial number of the wall where he does not want to arrange parking stalls.
- 3) Parking orientation : User enter 1 or 0 to choose a direction of internal parking.

Evaluator

- 1) Parking stall's number : minimum requirement from basement design review table

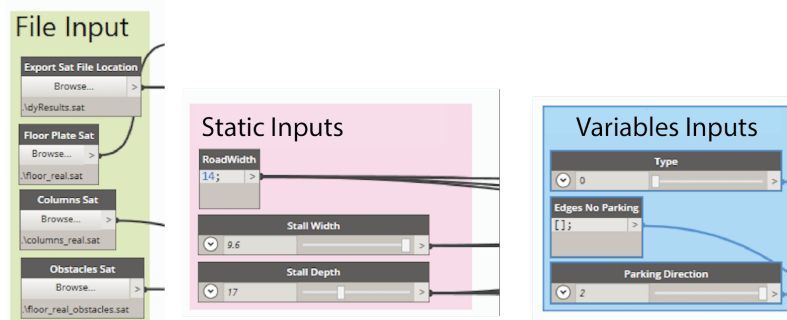


Fig 23. The UML for the simplest parking garage generative design prototype.

Step 4: Build up the parametric modeling scripts for generating the entire layout

Before building up the scripts for the entire floor plate generation, two minor functions were that can help architects improve basement modeling efficiency were developed first: function of automatic stall's location adjustment and function of fast place parking stalls along the road.

The first function can automatically adjust the location of stalls when the related column is moved. The second function can automatically place the stalls on the side of the road when architects draw a path on the center of the road. Architects can choose which side to place the stalls, and the stalls that generated by this function will avoid any conflict with the existing columns or obstacles.

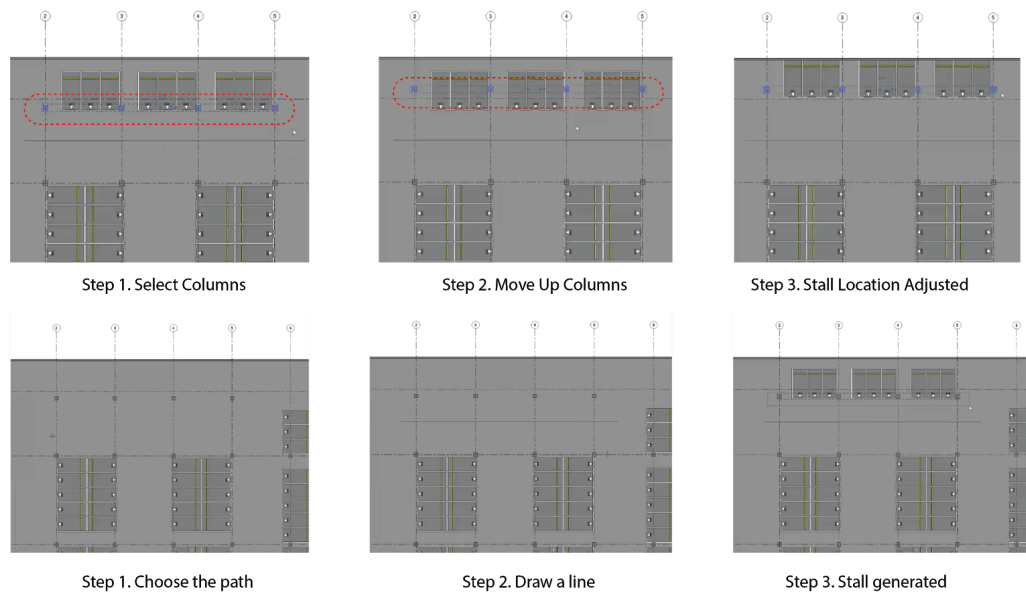


Fig 24. Top images show the function of stalls adjusting its location automatically when the location of columns changes. Bottom images show the function of using one traffic lane to place the stalls on the side.

Function to generate the entire basement layout

The complete script consists of customized dynamo components which are coded in C# and existing dynamo components from its library. The script will generate an entire basement parking layout in 2 steps. Step one is to generate parking stalls that are next to the boundary walls. Step two is to generate the internal parking stalls. Users are required to click the 'run' button twice in order to complete the layout generation.

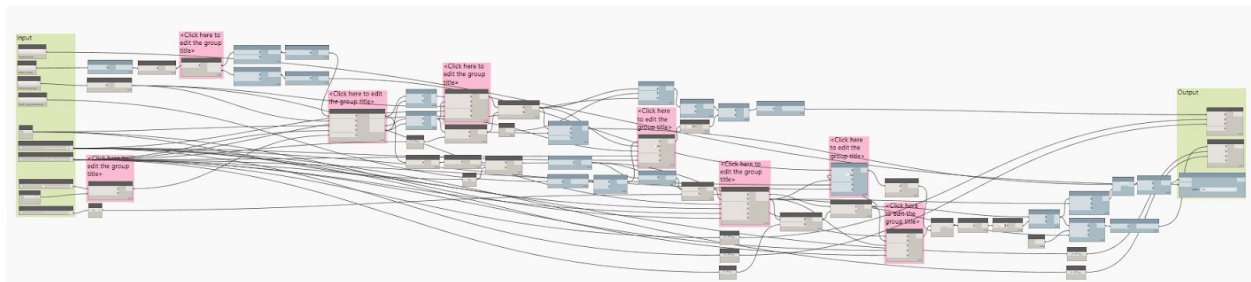


Fig 25. The complete prototype's scripts.

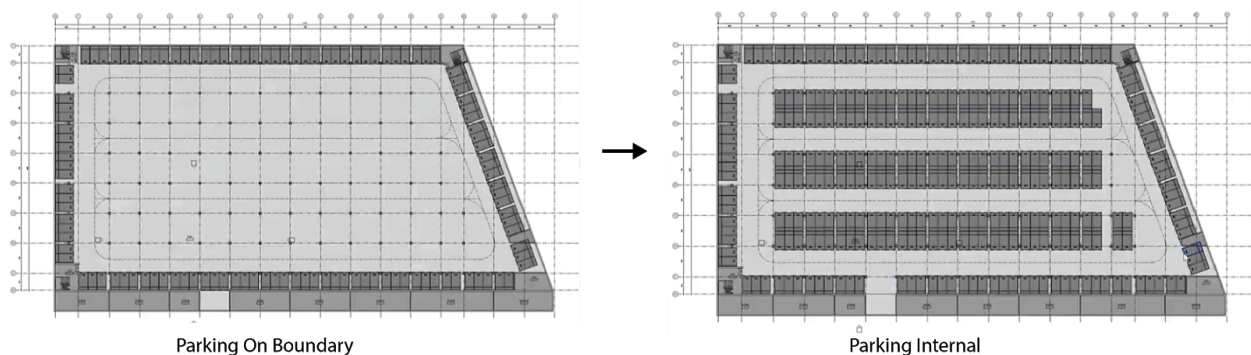


Fig 26. The basement parking layout generating process.

Function to optimize the generated layouts.

The last step is to connect the current scripts to the Project Refinery, which is a well encapsulated evolutionary solver developed by Autodesk. All we need to do is to connect the variable inputs to the plugins, set up the fittest function to maximize because we want to maximise the stall's number, set up the iteration round at 400. Then we can move to the prototype testing procedure.

4.4 Test the prototype with practical basement projects

As we defined in the development process, there are two dimensions we need to test: the ability of the software to restore a design solution, and the ability of the software to explore new solutions.

In order to test the first ability, I prepared a completed basement revit model and invited 4 architectural students as participants. I asked each of them to rebuild the model twice according to the model I gave. The first round they are required to only use the GD software. The second round they are asked to rebuild the model with revit. I'll record the time difference between the rounds and the model completion rate of the first round..

According to the results, the solutions generated in the first round can reach about 86.95% accuracy on average. There are 12 stalls out of 138 stalls that can't be modeled by the prototype, these stalls are divided into two cases: the stalls are parallel to the wall, and the stalls are occupying part of the lane. For the first case, considering that this way of parking is relatively rare, I had not completed the development of this feature at that time. For the second case, I think we shouldn't take it into consideration, because this extremely rare case should be individually adjusted by the architects later.

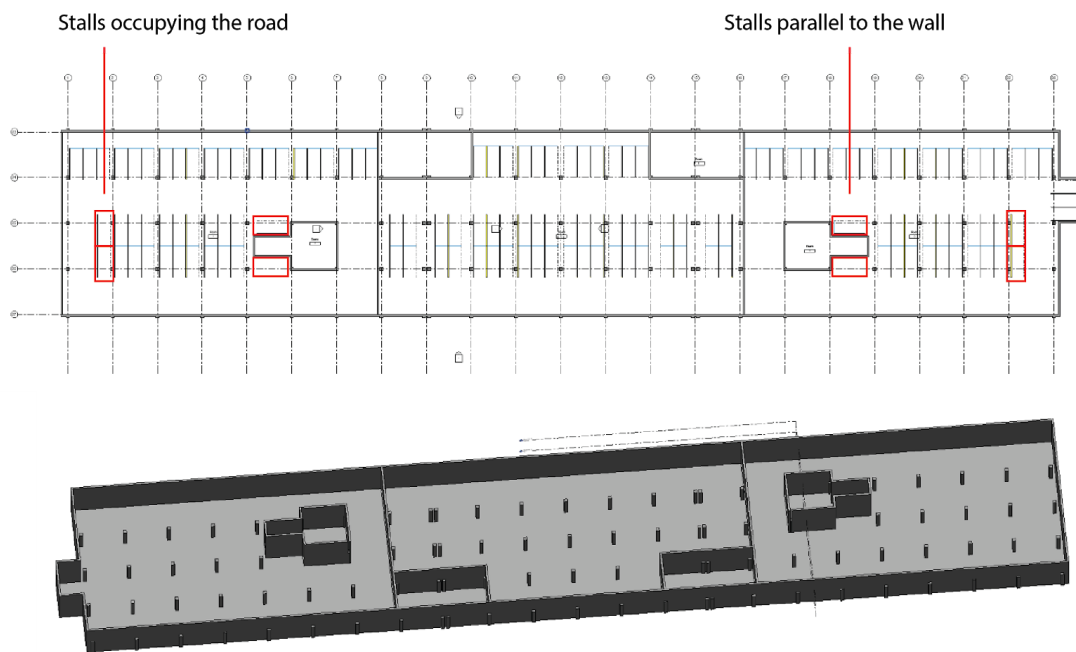


Fig 27. The basement model to test the ability of the prototype on restoring an existing design.

These students spent about 64% modeling time in the first round compared to the manual modeling round. The main reason they spent more time than expected is that the parking lot has three fire compartments. They need to prepare all the static inputs for each compartment. They complained that this step takes a lot of time.

In order to test the second ability, I prepared an empty basement revit model and invited 2 of my former colleagues to participate in the test. I give each of them 5 mins to explore the layout configuration of the given site, meanwhile, I use the GD tool to explore the layout. According to the results, two of the generated results produced 5% extra parking stalls compared to the one done by architects.

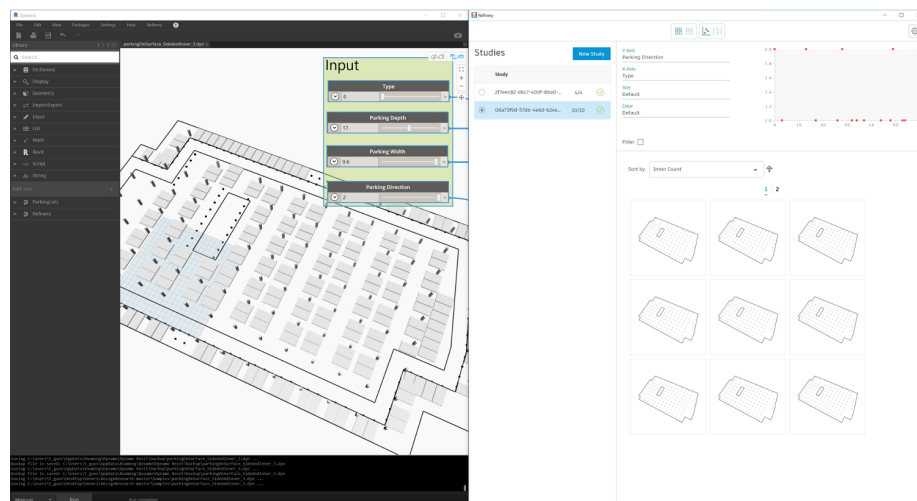


Fig 28. Run the project refinery to search for design alternatives.

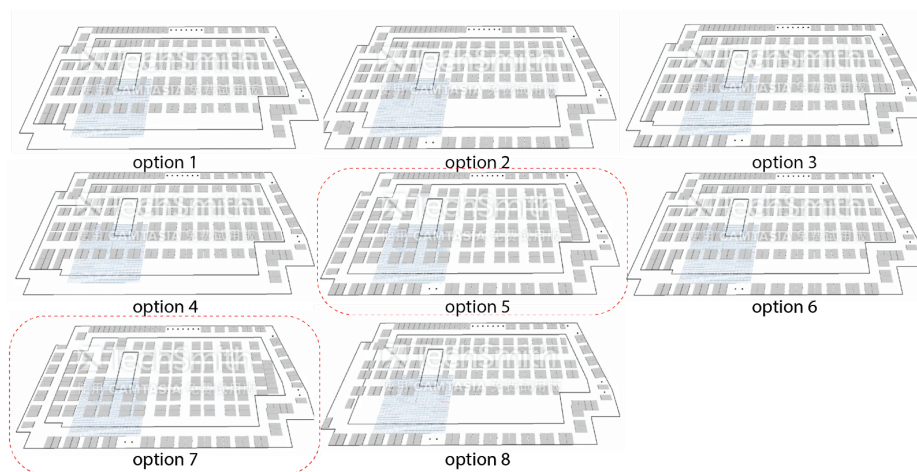


Fig 29. Part of the results generated by the refinery project, option 5 and 7 have maximized stall's number (n = 20 rounds).

After these two rounds of testing, all participants showed great interest in this prototype and asked to participate in the next version of the prototype. The results of these two rounds of testing also led me to find out the focus of the next step: to provide software with the function of dividing fire compartments and the function of placing stalls parallel to the walls.

Chapter 5. Discussion

In the last chapter, I summarized the main contributions I have made through the thesis exploration and the current limitations of the new development process I proposed. Finally, I discussed the potential future works we can do to overcome the limitations.

5.1 Contribution

In this thesis, I proposed a GD software process to guide developers to develop more effective GD software for architects. To demonstrate the feasibility of the process, I developed a GD prototype for a basement parking layout design task on dynamo. All the related dynamo scripts and C# codes are shared on a github repository that could be downloaded and used as a code template for similar GD software projects.

5.2 Limitation

There are three major limitations for the current proposed process:

- 1) **The process only works for architectural layout design tasks.** This is because the generative methods recommended by the process are mainly used for layout design problems. In addition, the dynamo scripts contained in the dynamo code template are also focused on handling different types of 2D geometrical problems.
- 2) **Lack of the latest technologies in the selection of the generative method.** The process currently only contains three mainstream generative methods for developers to choose. However, there are more and more GD software based on machine learning and reinforcement learning. For example the House-GAN project, which is a novel graph-constrained house layout generator based on a relational generative adversarial network (Nauata et al. 2020), and the GAN based room layout generator proposed in Stanislas Chaillou's thesis (Chaillou 2019). We need to gradually add these technologies into the methods selection chart.
- 3) **Not tested enough in practice.** So far we have only developed one GD prototype to test the feasibility of the process. In order to prove the effectiveness of the process and find

out potential problems, we should develop more GD software for different layout problems.

5.3 Next Step

There are some potential steps we can do to further improve the process:

First, we should build up more prototypes to keep testing the feasibility of the process. One idea I have in mind is that we can build a rule-based modeling GD prototype that handles the fire compartment division in the given basement. We can also integrate these two prototypes to make it as one software that is capable of producing a complete basement layout design.

Second, we could explore more generative methods including machine learning algorithms and reinforce learning algorithms. We should summarize the suitable application scenarios of these two technologies and add them into the method recommendation chart.

Third, we could also explore the possibility of modifying the process to make it fit for different fields of architectural design tasks such as facade form finding, architectural programming, and material selection.

In sum, this experience made me feel that this is a very interesting and promising field. I can imagine in the near future, the degree of automation of the architectural design process will be much better. The technological advances will liberate the productivity of architects and push the industry forward.

Reference

Chaillou, Stanislas. 2019. “AI + Architecture | Towards a New Approach.”

Coyne, Robert F. 1989. *Planning in Design Synthesis: Abstraction-Based LOOS (ABLOOS)*.

Davis, Daniel. 2019. “What ‘The Future of The Professions’ Reveals About the Future of Architecture.” 2019. <https://www.danieldavis.com/future-of-the-professions/>.

Davis, Davis. 2020. “Generative Design Is Doomed to Fail.” 2020. <https://www.danieldavis.com/generative-design-doomed-to-fail/>.

F Robert, Coyne. 1993. *Behavior Modeling in Design System Development*. Pittsburgh, Pa: Carnegie Mellon University].

Flemming, Ulrich, Robert Coyne, and T Glavin. 1987. “ROOSI - Version One of a Generative Expert System for the Design of Building Layouts,” 10.

Flemming, Ulrich, Halil Erhan, and Ipek Özkaya. 2004. “Object-Oriented Application Development in CAD: A Graduate Course.” *Automation in Construction* 13 (2): 147–58.

Khalili Araghi, Salman, and Rudi Stouffs. 2015. “Exploring Cellular Automata for High Density Residential Building Form Generation.” *Automation in Construction* 49 (January): 152–62.

Koning, H., and J. Eizenberg. 1981. “The Language of the Prairie: Frank Lloyd Wright’s Prairie Houses.” *Environment and Planning B: Planning and Design* 8: 295–323.

Krishnamurti, Ramesh, and Pedro Veloso. 2019. “62706 Generative System for Design.”

- Nauata, Nelson, Kai-Hung Chang, Chin-Yi Cheng, Greg Mori, and Yasutaka Furukawa. 2020. "House-GAN: Relational Generative Adversarial Networks for Graph-Constrained House Layout Generation." *ArXiv:2003.06988 [Cs]*, March.
- RAO, AJITH, and FILIZ OZEL. 2004. "An Evolutionary System to Generate Multi-Objective Architectural Design Solutions." In *Generative CAD Systems: Proceedings of GCAD 04, International Symposium on Generative CAD Systems*, by Ömer Akin and Ramesh Krishnamurti, 461–72. Pittsburg, Pennsylvania, USA: Carnegie Mellon University, School of Architecture.
- Smith, Lilli, Graceline Williams, and Mark Thorley. 2019. "Architectural Workflows: Office Layout." *Generative Design Primer* (blog). 2019.
- Stiny, G, and W J Mitchell. 1978. "The Palladian Grammar." *Environment and Planning B: Planning and Design* 5: 5–18. 1980. "The Grammar of Paradise: On the Generation of Mughul Gardens." *Environment and Planning B: Planning and Design* 7: 209–26.
- Stiny, George, and James Gips. 1972. "Shape Grammars and the Generative Specification of Painting and Sculpture." In , 1460–65. Amsterdam: North-Holland: C V Freiman (ed.) Information Processing 71.
- Veloso, Pedro, and Ramesh Krishnamurti. 2019. "From Black Box to Generative System." In *107th ACSA Annual Meeting Proceedings, Black Box*, 525–33. ACSA Press.
- Xiong, Dao hui. 2017. "Poly Real Estate -- Basement Research and Analysis."

Appendix

Project code, dynamo scripts, revit models

Project code, dynamo scripts, revit models can be accessed at:

<https://github.com/SIYUGUO/SiyuThesis>

Summary of interview question and findings

Below are the interview questions that I asked the participants to answer in online questionnaire and group session interviews:

Questionnaire:

Q1 Which country do you currently live in?

87.9% participants from the U.S, 6.1% participants from China, 6% from the other countries.

Q2 How many years of work experience do you have?

58.6% participants under 2 years of experience. 34.5% of participants have 3 to 7 years of work experience. The rest of the participants have more than 8 years of work experience.

Q3 Which of the following categories that you think you will be interested in use generative design tools in your current workflow? Please check your top three answers.

60% of participants voted for facade studies. 49.1% of participants voted for basement layout design. 38.2% of participants voted for program layout. 32.7% of participants voted for stair configuration.

Q4 Are there any other categories that you would like to try generative design tools? Please write it in the text box below.

The mentioned categories are : core layout, sustainable design and circuits layout.

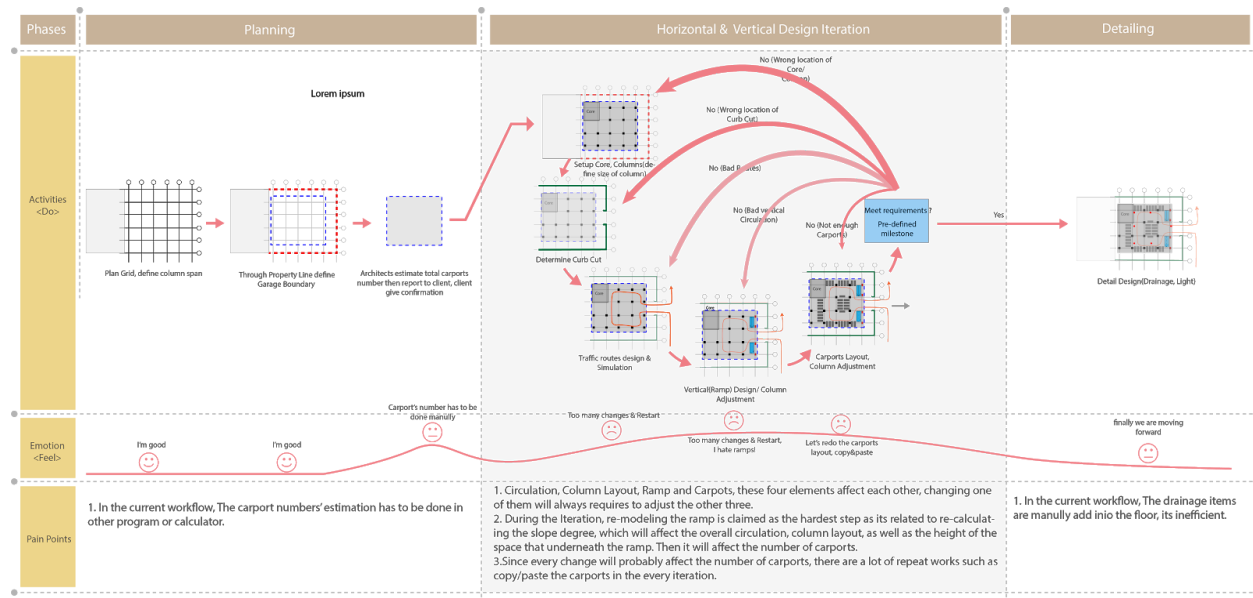
Group session interview questions:

Q1 When was the last time you worked on basement design?

More than 80% of participants have worked on basement design in the last two months.

Q2 Can you recall the entire process step-by-step?

Below is the user-journey map of the basement design process I've made after the interview.



In general there are three major phases : 1.size and grid planning, which include defining the boundary and column grids. 2. horizontal and vertical transportation design, which include design the location of entrance and exits, ramp, all the stalls, and the entire traffic lane. 3. detailing which includes fire facilities, MEP spaces etc.

Q3 Which step did you think was the most painful step during the process?

More than 50% of participants have mentioned modeling stall's modeling in revit. More than 60% of participants mentioned the ramp modeling in revit. The rest areas that have been mentioned by more than 30% of the participants are fire compartment design and circulation design.