'GHShot': a collaborative and distributed visual version control for Grasshopper parametric programming

Verina Cristie[1], Sam Conrad Joyce[2]
[1,2]Singapore University of Technology and Design
[1]verina_cristie@mymail.sutd.edu.sg [2]sam_joyce@sutd.edu.sg

When working with parametric models, architects typically focus on using rather structuring them (Woodbury, 2010). As a result, increasing design complexity typically means a convoluted parametric model, amplifying known problems: `hard to understand, modify, share and reuse' (Smith 2007; Davis 2011). This practice is in contrast with conventional software-programming where programmers are known to meticulously document and structure their code with versioning tool. In this paper, we argue that versioning tools could help to manage parametric modelling complexity, as it has been showing with software counterparts. Four key features of version control: committing, differentiating, branching, and merging, and how they could be implemented in a parametric design practice are discussed. Initial user test sessions with 5 student designers using GHShot Grasshopper version control plugin (Cristie and Joyce 2018, 2017) revealed that the plugin is useful to record and overview design progression, share model, and provide a fallback mechanism.

Keywords: *Version Control, Parametric Design, Collaborative Design, Design Exploration*

INTRODUCTION

The field of architecture is traditionally interdisciplinary and has adopted many technologies; from new materials in the industrial age, to software and algorithmic approaches in the digital era. CAD systems were first proposed in 1960s; and to further manipulate geometries, associative parametric programming was introduced in 1990s. This graph-based interface was popularised by tools such as Generative Components, Grasshopper, and Dynamo, where instead of using conventional text-based programming, visual programming is used.

Parametric modelling and software development bear similarities, particularly in their workflow and development cycle. Code produces software, while parametric modelling produces geometry. Hence, manipulating code results in change in software output, much like manipulating parameter and links between component results in change in geometry. Both software and parametric models are developed through iteration cycles. Modern software development practices 'Agile' method (Huo et. al, 2004) where code is developed in iterative cycle of building and testing; a parallel to an iterative process of design solution generation and evaluation.

Complexity in Software Development and Parametric Modelling

With the rise in use and complexity in programming, similar challenges have been faced managing this in both fields: software code reached millions of lines, and equally parametric models to many hundreds of components and thousands of links in commercial projects. Tangled links or 'spaghetti' parametric model (see Fig. 1) is a classic example of unmanageable models (Davis et. al, 2011). Large parametric models are often inflexible (simple change often breaks models); model's changes are often not easily found/detectable, and model reuse and sharing problematic (Smith, 2007).

Cheaper and more powerful computers in the 1960s allowed for wider use of software, and larger more complex software requirements, driving the software industry into the so-called 'software crisis' (Dijkstra, 1972). Software crisis is identified as incomplete and degrading software performance due to unmaintainable software complexity (Valdez, 1988). The most popular case was the decade long IBM 360 Operating System development between 1960 and 1970 that resulted in a multi-million-dollar project overrun; the reaction to which catalysed a structured transformation of the software development process to software engineering (Brooks, 1995). Key good practices resulting were object-oriented programming (OOP) and source code control system (SCCS) (Rochkind, 1972) were both developed during this period of time. Both of which will be further discussed in later parts of this paper, but these combined user-best-practices and technical solutions resulted in the ubiquitous software development approach we have today; where it is not uncommon to have tens to even thousands of people collaborating on code bases millions of lines long to build scalable complex software projects from the '80s onwards. However, in architecture, this structured usage focus has been mainly on CAD or BIM model complexity and only recently has parametric modelling begun to exhibit acute issues in similar ways. For example, in March 2019, Autograph[1] plugin was released to help automatic arrangements of objects in Grasshopper canvas, signifying growing complexity in parametric modelling. A thread in Grasshopper forum [2] in April 2018 showed large grasshopper files designers generate, ranging from 1000 to even 20,000 components. Hence, there is a growing critical drive to start looking into adopting not only the technologies in software design, but also the good practices of the software development process to parametric design.

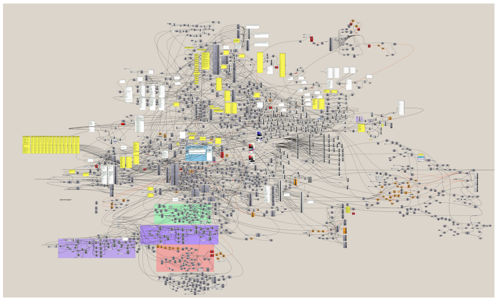


Figure 1
Grasshopper model with 1,234 components (Picture credit: Lukasz Domagala)

Adoption of Object-Oriented Programming (OOP)

In OOP practice, programming complexity is reduced by splitting up code into functional reusable manageable pieces called 'objects'; i.e.: to encapsulate data and functions that operate on it into modules. Modularity and encapsulation are two principles of OOP. Which uses nouns to explain components/data, and verbs to explain function action. This makes it clearer what is happening within code; reducing mental overhead whilst programming. Davis (2013) recommended parametric models to adopt this modularity to address inflexibility. An inflexible parametric model breaks or needs substantial remodelling even for minor changes, as lack of hierarchy and separation of variables and functions means a lot of interconnected networks. By partitioning parametric models into separate logical elements according to their functions, models are easier to understand, edit, and reuse; even as design develops, especially in the case of later contributors. Pena (2014), further emphasized that with separation of functionality, parametric models could use internal and external code to communicate and extend the functionality of the model, much like the use of different libraries in software. A recent example of this OOP practice can be seen in a 'distributed data model' (Wortmann and Tunçer, 2017) paradigm used in the Morpheus Hotel project (Muscettola et al, 2017), where the model was divided into hundreds of parametric model and geometry files, of which each is processed individually before combining them again, to manage its 1,668,301 different parts of building structures and façade systems.

Next: Adoption of Version Control System?

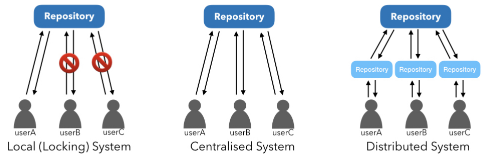


Figure 2
Three generations of code versioning system: local (locking) system (left), centralised system (middle), and distributed system (right).

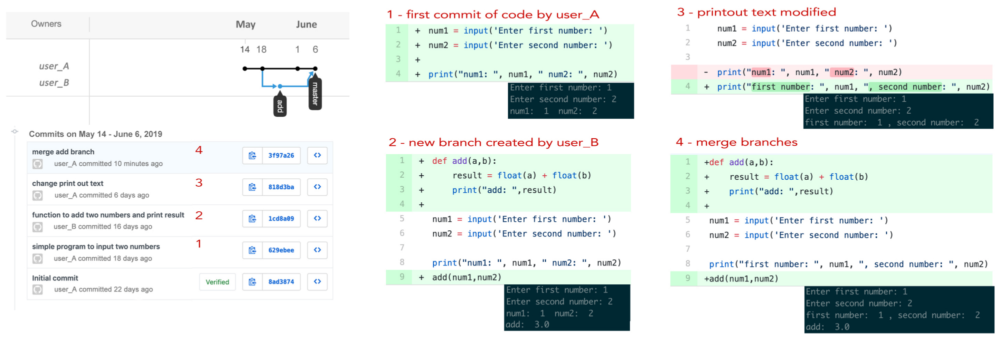
While OOP practice has a growing adoption and positive impact specifically in organised commercial parametric modelling practice, SCCS (more commonly called version control systems) are so not well known and even less so applied in parametric design circles. In software development, version control is now almost ubiquitously used to maintain different versions of code and collaboration.

The fundamental concept of versioning revolves around code saving (pushing) and code downloading (pulling) to and from a shared repository between collaborators. There are three types of versioning systems (see Fig. 2): local (locking), centralised, and distributed. In a locking system, many versions could be created but only one user is allowed to access code at a time (only one channel of push and pull). In a centralised system, everyone can pull code at any time, but before pushing to the repository, one should always have the latest version. If currently user A and user B are editing the same code, and user A pushes his/her edit first, user B will need to first download that edit before able to push his/her own edit. In this manner, a single central latest version is always maintained. Lastly, in a distributed system, each machine has a local repository and manages its own versioning of code. Users can make local checkpoints ('commits') before sending ('pushing') them to the server, and multiple files of the same version can exist. Users only need to merge them when they choose to do so.

Locking system is the earliest versioning system developed and in practice, it is rarely used as it doesn't allow collaborators to work in parallel. Aish (2000) proposed centralised versioning system for design in Bentley's Digital Project (currently named as Project Wise) as there is a need for a clear record of action - who did what in the project. Comparably, Burry and Holzer (2009) also implemented a centralised sharing system to allow ease of parametric model sharing in Gehry Technologies. However, the project had an issue with the binary file merging and thus in its implementation, a locking system was used instead. Both implementations, nevertheless, were BIM-based, and more apt for the later stage of design; and as such could not support creative process well.

PARAMETRIC MODEL VERSIONING

'GHShot' was developed to realise a distributed versioning system for parametric modelling in Grasshopper (Cristie and Joyce 2018, 2017). It consists of (1) a custom Grasshopper component developed to allow users to send design progression directly from their modelling environment, and (2) a cloud repository to store the sent parametric models and its attributes. This cloud server provides webpage views for users and others to view and navigate projects and designs, as well as give feedback and download models. In the next part, common practices in distributed versioning systems - namely committing, branching, differentiating, and merging code, and what these practices mean for parametric modelling, and how it could be translated into parametric modelling practice will be further discussed.



Commits on May 14 - June 6, 2019

merge add branch 4 3f97a26
user_A committed 10 minutes ago

change print out text 3 818d3ba
user_A committed 6 days ago

function to add two numbers and print result 2 1cd8a09
user_B committed 16 days ago

simple program to input two numbers 1 629ebee
user_A committed 18 days ago

Initial commit Verified 8ad3874
user_A committed 22 days ago

1 - first commit of code by user_A

```
+ num1 = input('Enter first number: ')
+ num2 = input('Enter second number: ')
+
+ print("num1: ", num1, " num2: ", num2)
```

```
Enter first number: 1
Enter second number: 2
num1:  1  num2:  2
```

2 - new branch created by user_B

```
+ def add(a,b):
+     result = float(a) + float(b)
+     print("add: ",result)
+
  num1 = input('Enter first number: ')
  num2 = input('Enter second number: ')

  print("num1: ", num1, " num2: ", num2)
+ add(num1,num2)
```

```
Enter first number: 1
Enter second number: 2
num1:  1  num2:  2
add:  3.0
```

3 - printout text modified

```
  num1 = input('Enter first number: ')
  num2 = input('Enter second number: ')

- print("num1: ", num1, " num2: ", num2)
+ print("first number: ", num1, ", second number: ", num2)
```

```
Enter first number: 1
Enter second number: 2
first number:  1 , second number:  2
```

4 - merge branches

```
+def add(a,b):
+    result = float(a) + float(b)
+    print("add: ",result)
+
 num1 = input('Enter first number: ')
 num2 = input('Enter second number: ')

 print("first number: ", num1, ", second number: ", num2)
+add(num1,num2)
```

```
Enter first number: 1
Enter second number: 2
first number:  1 , second number:  2
add:  3.0
```

Figure 3
Figure 3 - Code versioning example of a simple addition program written by two programmers. User_A started by writing code that will take two numbers and print them (1). User_B wrote add function to add both numbers by branching the code (2) and merge it back (4). User_A also modified part of the code (3). Action records are historically tracked (left).

Parametric Model Development Progression Capture (Committing)

In software versioning, when code is committed a checkpoint is created, an internal 'save as' functionality so that programmers don't have to create a naming convention system/structure for each commit. Instead, the programmer creates a commit message, which is a message describing the changes in the current commit. This is helpful for the code owners themselves and other programmers who look at the code as they will be able to understand what intentions are of the changes in that specific commit. Once the code is committed, the owner then can push the code to the server so that the code is saved in the repository. By periodically pushing the changes all collaborators who have access to the cloud can understand the progression of the code and pull to work on their own versions based on someone else's code. In addition, should the code break in a way that can't be understood or fixed, a previous commit can be pulled where the state of the code worked before and developed from here.

Meanwhile, without a versioning system, generally files of different versions are maintained by making copies and giving each file a different name. For example, designers making modification to their designs could name their Grasshopper files: file_1.gh, file_2_v2.gh, file_3_finalfinal.gh, and so on. This irregular naming convention failed to describe relationships between the files (e.g. how is file_1 related to file_3_finalfinal?). On the other hand, while in a typical BIM environment, files are strictly named - coding everything from project code, level, location, classification, etc (see Fig. 4), this system is unwieldy and difficult to enforce in the early design stage. Additionally, it also lacks visual representation of each file that could represent the design progression better. Temporal design versions per designers' commit are especially important to monitor design evolution, identify individual contributions, and combine and reuse concepts (Aish, 2000). Work by Sakai and Tsunoda (2015) demonstrated how a history tree can be used to track collaborative house design progression visually in 3D WebGL. This is a good starting point towards a better design history documentation and managing design complexity; despite the limitation of 3D web-based geometry editor for producing designs. GHShot expands on these ideas to bring a better design versioning practice to the wider community of parametric designers with the commonly used Grasshopper.

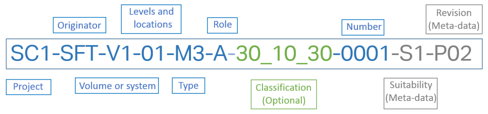


Figure 4
BIM System naming convention (Source: [3])

GHShot implemented a distributed versioning system as it is important for designers to work on their own versions during exploration stage and not be confined to the central master version. However, unlike traditional distributed system that has both local and cloud repository, local repository was not implemented. Parametric modelling is mostly done on a workstation connected to an online system unlike programming's remote work. Hence, commit and push operations are combined into a save operation on the GHShot Grasshopper component button. This was also chosen remove overhead of designers having to learn another local versioning system. Upon clicking, the parametric model and its attributes will be sent to the URL set and designers can immediately see what they have just sent in the web. Users can commit their designs at any relevant point during the design progression, and other designers can download the models and modify them. Designs committed are shown in a tree visualisation on the web so that design progression can be traced.

Parametric Model Change Detection (Differentiating)
In software versioning the 'Diff' (short for difference) utility tool (Hunt and McIlroy, 1976) and its many derivatives is a popular tool to compare and display differences between two files. By being able to understand what changed between an old and a new version, programmers are able to identify lines of code that made up the new features in the software, and/or contributed to a new error in the software. Given that many versions/options of design are saved, a parametric 'diff' would be helpful to understand the change in between the parametric model - such as what components and links are deleted, added, or changed, and designers could potentially see how that change relates to the geometric difference.

To show changes, Diff tool highlights deleted lines of code from the old version in red and added lines in the new version in green. In the tool 'MACE', Zaman et. al (2017) focused on making an interface that could highlight differences of nodes (components) and edges (links) between different design alternatives. However, the tool is built on top of a stand-alone 2D parametric pattern generator tool and doesn't provide end-to-end support for parametric modelling process. In GHShot, changed components and links in between commits are highlighted in red (deleted), green (added), and yellow (changed attributes inside components) for a downloaded model.

Parametric Model Alternative Generation (Branching)
Both parametric modelling and software suffer from the challenge of unclear goals in the beginning and changing goals throughout development. For example, a new software feature could suddenly be requested while the software has already been used for the client's daily operation. A versioning system facilitates this by allowing programmers to write the code for the new feature in a 'branch'. By doing so, this new feature could be tested separately and would not disrupt the master (stable) code used currently. Similarly, programmers could also develop different features in different branches so they would not disrupt each other's work.

In parametric modelling environment, we believe this branching practice embodies two types of design activities, both resulting in creation of design alternatives: (1) exploring a different but parallel design direction and (2) exploring a range of comparable design options within a parametric exploration (changing parameters). Generation of alternatives is a core activity in parametric modelling, and this should be done seamlessly in any versioning system for these systems. In GHShot implementation, True/-False Toggle is used. A designer can choose to set as true or false while committing the design. Setting to true means that a new design concept/options is generated, while setting to false means a normal commit of progression and not branching.

Parametric Design Convergence from multiple alternatives (Merging)

In software practice features or fixes done on code branches are eventually merged back into the master code if they are useful or abandoned if they are deemed not useful. However, in the early parametric design phase there is unlikely to be a 'master' version yet. As many design alternatives are explored, design branches could be created. Merging practice then could be seen as making a new design iteration inspired by previously created (two or more) alternatives.

Software code is text-based and automatic merge from two different branches could be done as long as there is no 'conflicts', or different code in the same line number. Grasshopper parametric model is XML-based, and hence is also text-based. However, simply adding components and links from one Grasshopper file to another could cause the parametric model not to work correctly. A better way of merging is then for a designer to copy parts from other files to his/her own file to further develop the design.

With parametric model versioning tool in place, parametric design progression is now captured, stored in the cloud repository. Parametric model 'snapshots' mean empirical data of who did what design, at what time. In the next part, what could be done with this data, and how this data could facilitate a design exploration process will be discussed.

BEYOND CODE AND PARAMETRIC MODEL VERSIONING

Analytics and User Feedback

Although Git [4] is a popular versioning tool, distributed versioning didn't gain its ubiquity and important position until the rise of Github [5]. Git is typically installed in programmer's local machine, and programmers can push and pull their code to and from Github cloud repository, essentially sharing their code to their teams or wider programmer community. As of May 2019 [6], there are over 39 million user accounts and over 29 million projects on Github. One important additional feature of Github on top of Git is code analytics; where an overview of the number of commits done by each user, code frequency, traffic of code, and other project-related information are given and visualised in graph format. Having this feature gives a team insight to monitor their project and streamline their processes. In addition, with the community-driven (public) repository, feedback or issues may be quickly gathered and acted upon, improving the software in faster iterations.

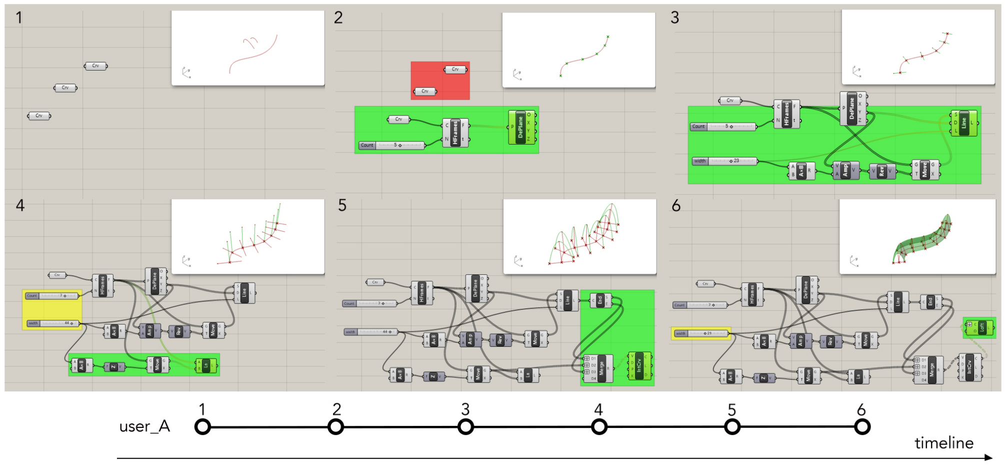


Figure 5
Parametric model commits and difference detection. From initial three existing lines (1), two lines are deleted (2) – highlighted in red. Length of horizontal and vertical intersecting lines are modified (4) – highlighted in yellow.

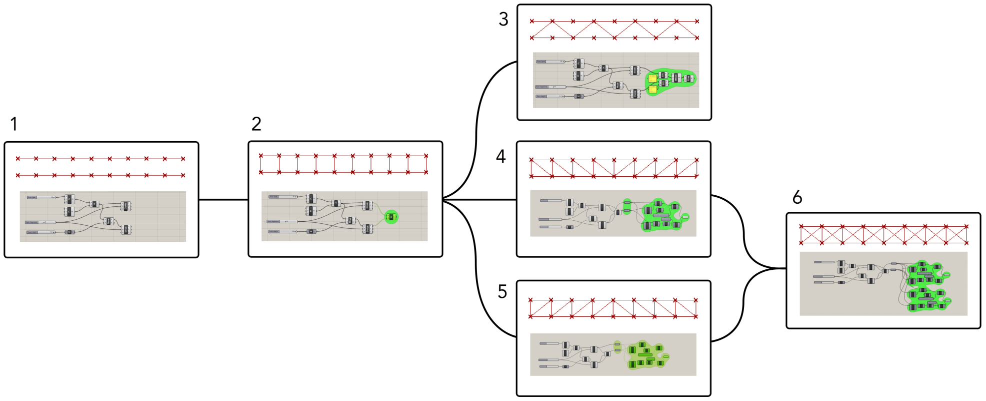


Figure 6
Parametric Model Branching and Merging Practice. Commit (2) has 3 branches (3,4,5) based on its diagonal pattern. Commit (6) is the merge from (4) and (5)

Inspired by features in the Github repository, the web repository of GHShot consists of four main views: (1) snapshot view (see Fig. 7), (2) history tree (see Fig. 8), (3) design analytics (see Fig. 9), and (4) 3D model view with commenting and rating feature (see Fig. 10). Snapshot view gives a quick overview of latest designs uploaded. The history tree provides relationship between designs, showing the project's parametric model progression and exploration. Analytics view allows comparison of quantitative or categorical performance attributes of design options. The 3D viewer enables users to give feedback on the design; this dual view is analogous to the duality of objectivity and subjectivity in design - there is no 'true' or 'false' solution, but 'good' or 'bad'. Design, after all is a 'wicked problem' (Rittel and Webber, 1973), and will stop when a satisfacing solution is found. Further, as a difference in code could produce a difference in software outcome and performance, different design schema produces different geometry and its performance output. Versioning parametric model allows progressive modelling (Wang et. al, 2019) - parametric models of different design schema can now be compared for optimisation effort.

GHShot User Test Session and Interview

To understand how designers would interact with a parametric design versioning tool, and how the tool could likely affect design exploration process, we conducted GHShot user test session followed by interview after. Five student designers (undergraduate and postgraduate) with 2-10 years of Grasshopper experience were to explore a simple tower structure parametric model (approximately 20 components) individually, making models of their liking while still minding the performance value (deflection and utilization). One-hour time limit was given, and GHShot was to be used in the process.

Table 1
Summary of designer's profile and GHShot use

	D1	D2	D3	D4	D5
Years with Grasshopper	2	6	10	2	2
GHShot's ease of use (1-10)	5	8	10	8	9
Number of commits	11	12	14	3	25
Number of branches	0	2	2	0	4

Designers were asked if GHShot was easy to use; as an easier tool means less learning curve and likelier for designers to integrate GHShot tool into their design workflow). Most designers found the tool to be easy to use (see Table 1), except D1 who gave 5/10 because of technical difficulties - the slow speed sending model from Grasshopper to the server. Designers were observed to be saving their progression periodically and creating branches as they made design alternatives (see Table 2). D1 and D4 were asked why they didn't make any branches, and it was because of the 1-hour time limit. As they did not save too many options, linearly saving was enough and consistent with their linear exploration. Should they be given a longer time and to collaborate with other designers, they would use the branching feature more. D5 asked if it was possible to modify the tree history (by deleting or moving designs). In this manner, GHShot was treated as a 'design gallery system' (Woodbury et al, 2017), where designers filter and manage design options they generated.

Below is feedback on GHShot extracted from the interview after the design session:

On branching and design history view:

- Helpful to understand the link between the geometry and parametric, to remind yourself why you change certain things (D1)
- Idea of making branches and versions is really easy to understand. Other people can look at your work and understand how you started at one point and came up with certain option (D3)

On analytical graph:

- Useful to compare designs (D1), and see trends (D5)
- Useful to see how change in elements affect performance (D2)

On 3D Viewer:

- Easily sending and viewing my model online means I could free up my machine's graphic resources (D4)
- Good to see progress, especially for collaboration or presentation (D5)

Others:

- Convenience of having backup copies should the current version doesn't work (D2)

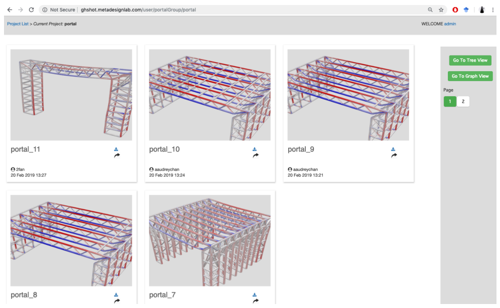


Figure 7
Snapshot View

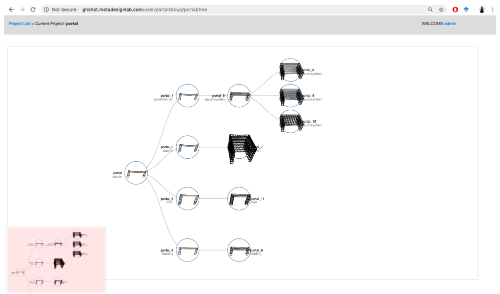


Figure 8
History Tree

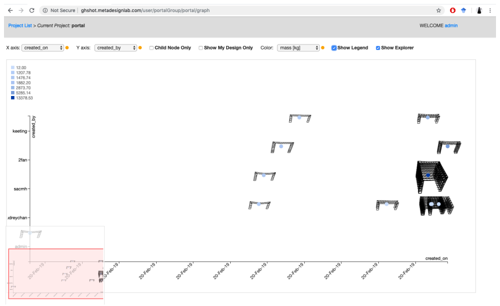
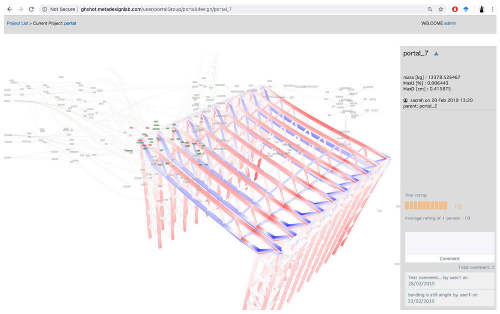


Figure 9
Design Analytics


ghshot.metadesignlab.com/user/portalGroup/portal/design/portal_7
Project List > Current Project: portal
WELCOME admin
portal_7
mass [kg] : 13378.526467
MaxU [%] : 0.006443
MaxD [cm] : 0.415875
sacmh on 20 Feb 2019 13:20
parent: portal_2
Your rating:
Comment
Figure 10
3D Viewer

- Unlike going back and forth with undo and redo, having different copies of your designs allow you to think about your overall design (D3)
- The tool allows not only to look at immediate previous design iteration, but also the overall design iteration from the beginning, allowing you to choose alternatives that might not be the best performance, but good performance enough to continue developing interesting design (D5)

Overall, the feedback received seemed to be positive, leveraging on the GHShot's capability to capture end-to-end design progression and design options as they are generated, and thus potentially giving designers the capability to navigate their exploration process better. Despite the individual user test session conducted, designers mentioned how the tool can be useful in a collaborative environment, which will be conducted in our future testing.

CLOSING

We started by laying out the similar pattern of growing complexity faced by the software development field and parametric modelling. Object-oriented programming (OOP) and code versioning system were born out of the need to improve collaboration and manage the complexity that comes with bigger and more complex software projects. Parametric modelling has started to adopt the paradigm of modularity - a concept of OOP, where there is a clear separation of functions inside parametric model, resulted in a more structured/organised design schema. In parallel to modularity, we are proving a powerful next step to take is for the design community to explore the potential adoption of a design versioning tool.

The current state-of-the-art distributed versioning system provided a clear picture of how a versioning system can be used as a design support tool during the exploration phase, especially in collaborative environments where options are generated in parallel. Four features of a versioning system: committing, differentiating, branching, and merging, had been discussed on how it could be implemented in a parametric modelling environment. A pilot study with GHShot Grasshopper plugin and 5 student designers revealed positive responses despite the short one-hour testing. Parametric modelling versioning system has the potential to provide a clear progression of design progress (especially for design documentation and presentation), to allow ease of sharing and thus faster feedback and iteration process in a collaborative environment. In additional, parametric model progression visualisation and analytics could give better insight and navigation of the whole design process.

Finally, we are reminded that both software design and architectural design are of different worlds. Architectural design often has ambiguity and needs interpretation (Vardouli, 2014). In this work, we essentially tap on the fact that parametric models share explicit externalisation (Aish and Woodbury, 2005) - a similar trait to source code. And hopefully, in the long run, with versioning system, 'transparency in information architecture' (Hirschberg, 2003) could be captured, and hence, in the long run, the empirical data (design) captured could contribute to design process improvement. Future works involve releasing our plugin to food4Rhino, a Grasshopper plugin repository, to receive feedback from the wider Grasshopper community.

REFERENCES

Aish, R 2000 ', Collaborative Design using Long Transactions and "Change Merge"', *Proceedings of eCAADe 18*

Aish, R and Woodbury, R 2005 'Multi-level interaction in parametric design', *International symposium on smart graphics*, pp. 151-162

Burry, J and Holzer, D 2009 '). Sharing design space: Remote concurrent shared parametric modeling', *Proceedings of the 27th eCAADe*, pp. 333-340

Cristie, V and Joyce, SC 2017 'Capturing And Visualising Parametric Design Flow Through Interactive Web Versioning Snapshots', *IASS Annual Symposium*, Hamburg, Germany

Cristie, V and Joyce, SC 2018 'GHShot: 3D Design Versioning for Learning and Collaboration in the Web', *Extended Abstracts of the 2018 CHI Conference on*

Human Factors in Computing Systems, Montreal, p. LBW107
Davis, D 2013, *Modelled on software engineering: Flexible parametric models in the practice of architecture.*, Ph.D. Thesis, RMIT
Davis, D, Burry, J and Burry, M 2011, 'Understanding visual scripts: Improving collaboration through modular programming', . *International Journal of Architectural Computing*, 9(4), pp. 361-375
Dijkstra, EW 1972, 'The humble programmer', *Commun. ACM*, 15(10), pp. 859-866
Hirschberg, U 2003, 'Transparency in information architecture: Enabling large scale creative collaboration in architectural education over the Internet', *International journal of architectural computing*, 1(1), pp. 12-22
Hunt, JW and McIlroy, MD 1976 'An algorithm for differential file comparison', *Bell Telephone Laboratories CSTR #41*
Huo, M, Verner, J and Babar, MA 2004 'Software quality and agile methods', *In Computer Software and Applications Conference*, pp. 520-525
Brooks Jr, FP 1995, *The Mythical Man-Month: Essays on Software Engineering*, Pearson Education India
Pena De Leon, A 2014, *Separation of concerns: strategies for complex parametric design modelling*, Ph.D. Thesis, RMIT
Muscettola, V, Salvi, M, Mutyaba, M, van der Heijden, R, Tai, A and Levelle, E 2017 'The Morpheus Hotel: From Design to Production', *www.rhino3d.com/go/morpheus*
Rittel, HW and Webber, MM 1973, 'Dilemmas in a general theory of planning', *Policy Sciences*, 4(2), pp. 155-169
Rochkind, MJ 1975, 'The source code control system', *IEEE transactions on Software Engineering*, 4, pp. 364-370
Sakai, Y and Tsunoda, D 2015 'Decentralized Version Control and Mass Collective Collaboration in design-A Case Study of a Web Application Utilizing the Diff Algorithm and Automated Design Generation', *Proceedings of eCAADe 33*, pp. 207-214
Smith, R 2007 'Technical Notes from experiences and studies in using Parametric and BIM architectural software', *http://www.vbtllc.com/images/VBTTechnicalNo tes.pdf*
Valdez, MEP 1988, *A gift from Pandora's box: The software crisis*, Ph.D. Thesis, University of Edinburgh
Vardouli, T and Buechley, L 2014 'Open source architecture: an exploration of source code and access in architectural design', *Leonardo 47(1)*, pp. 51-55
Verina, C and Joyce, SC 2018 'GHShot: 3D Design Versioning for Learning and Collaboration in the Web', *Extended Abstracts of the 2018 CHI Conference*
Wang, L, Janssen, P and Ji, G 2019 'Progressive Modelling for Parametric Design Optimization', *Proceedings of the 24th CAADRIA*, pp. 400-409
Woodbury, R 2010, *Elements of Parametric Design*, Routledge
Woodbury, R, Mohiuddin, A, Cichy, M and Mueller, V 2017, 'Interactive design galleries: A general approach to interacting with design alternatives', *Design Studies*, 52, pp. 40-72
Wortmann, T and Tuncer, B 2017, 'Differentiating parametric design: Digital workflows in contemporary architecture and construction', *Design Studies*, 52, pp. 173-197
Zaman, L, Stuerzlinger, W and Neugebauer, C 2017 'MACE: A New Interface for Comparing and Editing of Multiple Alternative Documents for Generative Design', *Proceedings of the 2017 ACM Symposium on Document Engineering*, pp. 67-76

[1] https://www.food4rhino.com/app/autograph
[2] https://discourse.mcneel.com/t/whats-your-largest-grasshopper-script-the-hall-of-shame/60594
[3] https://bimportal.scottishfuturestrust.org.uk/level1/stage/8/task/47
[4] https://git-scm.com/
[5] https://github.com/
[6] https://github.com/search?l=&o=desc&q=followers:%3E-1&ref=advsearch&s=joined&type=Users