P. Janssen, P. Loh, A. Raonic, M. A. Schnabel (eds.), *Protocols, Flows and Glitches, Proceedings of the 22nd International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA) 2017*, 613-623.

COLLABORATIVE ALGORITHMIC-BASED BUILDING INFORMATION MODELLING

SOFIA FEIST[1], BRUNO FERREIRA[2] and ANTÓNIO LEITÃO[3]
[1,2,3]*INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal*
[1,2]*{sofia.feist|bruno.b.ferreira}@tecnico.ulisboa.pt*
[3]*antonio.menezes.leitao@ist.utl.pt*

Abstract. Algorithmic-based Building Information Modelling (A-BIM) allows the development of BIM models through algorithms. In a collaborative environment, A-BIM requires management strategies to deal with concurrent development of architectural projects. However, despite there being several tools that support this type of collaborative work, they are not appropriate for A-BIM because: (1) they track changes in the generated model instead of the code where the changes originate from, and (2) they are vendor-specific while A-BIM models might be generated for different BIM applications. In this paper, we discuss the use of Version Control (VC) for project management and concurrent development of A-BIM projects. We evaluate VC for A-BIM through a series of scenarios in the context of a case study.

Keywords. Algorithmic Design; Programming; Algorithmic-based Building Information Modelling; Version Control; Collaborative Design.

1. Introduction

Over the years, the merits of Algorithmic Design (AD) have been extensively researched in the Architecture, Engineering and Construction industries, from the integrated design and fabrication of complex freeform geometries (Kolarevic 2003; Pottmann et al. 2015) to the automation of repetitive, time-consuming tasks and processes (Burry 2011; Santos et al. 2012).

However, despite having been explored with Computer-Aided Design, AD has only recently started to be addressed in the context of Building Information Modelling (BIM). With BIM, an integrated database of building information is developed collaboratively amongst the different members of the project team in order to digitally simulate the planning, design, construction and operation of a building (Azhar et al. 2009; Eastman et al. 2008).

In the past, an approach named *Algorithmic-based Building Information Modelling* (A-BIM) was proposed, which combines AD with BIM (Feist et al. 2016). A-BIM is the use of algorithms to generate BIM models, which are indistinguishable from manually generated ones. However, instead of having the user providing the building information directly to the BIM model, in A-BIM this information is provided by the algorithm that produces the model, using primitive operations that create building components. This approach can also be used to generate identical models in different BIM applications, thus reducing interoperability issues which cripple collaborative practices.

Like traditional BIM, A-BIM also benefits from collaborative development processes. However, A-BIM requires different management strategies to handle the inputs produced by the multiple participants. Issues such as the concurrent development of programs, the coordination between individuals, and management of information, are important challenges that must be taken into account.

When using CAD and BIM applications in the traditional manual way, development teams can use tools, such as Autodesk's Vault and Bentley's ProjectBank, to facilitate collaborative work, allowing team members to develop the model in parallel and track changes made by other team members. However, these tools are not appropriate for algorithmic approaches such as A-BIM, since changes need to be tracked in the code, instead of the model, and models might be generated in multiple tools.

Version control (VC) systems (Brindescu et al. 2014) can offer a solution to these limitations. These systems have been used in Computer Science for many years, which has led to the development of several tools, such as Concurrent Versions System (CVS) and Git. Due to the algorithmic nature of A-BIM, these tools might be more appropriate than the ones traditionally used with CAD and BIM applications, making differences in code easier to detect and manage. In this paper, we explain VC and discuss its potential to manage the development of collaborative AD projects, specifically in the case of A-BIM. Afterwards, we discuss and analyse the application of VC through a series of scenarios found in the development of a collaborative case study.

2. Version Control

Collaboration in software development is supported by a variety of tools, among which VC systems. VC helps programmers control and manage changes introduced by multiple developers. By storing every change made by every programmer involved, VC preserves the project's history and allows individual changes to be identified, along with their author, date and purpose. This enables developers to go back to earlier versions of the project to check, modify, or revert changes introduced earlier.

VC also supports independent, concurrent work through *Branching* and *Merging*. Branching is used to pursue independent lines of development in the project, e.g. a new feature for the program. By isolating different lines of development in separate branches, developers can work independently on separate changes without affecting others, until they are ready to share those changes with the main line

of development. Branching also ensures that unstable code is never introduced into the main line. Once independent changes are ready to be integrated, developers can merge their branches back into the main line (figure 1). Merging combines changes from multiples branches into a single one. If changes are compatible, e.g., if two completely different parts of the program were modified, then they can be automatically merged. On the other hand, if changes are incompatible, the developers are informed about the conflict and have to manually solve it. Conflict detection in VC systems is based in the textual comparison of different versions, which means that these systems are more appropriate for textual programming languages.

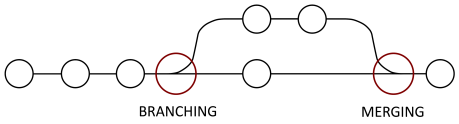

Figure 1. Merging process between two branches. Black circles represent recorded instances in the development of the project where changes were introduced.

VC systems can be either centralized or distributed (figure 2). While Centralized Version Control (CVC) stores the project's information in a single central repository, with Distributed Version Control (DVC) each developer gets their own local repository with all of the project's information. With CVC, in order to make changes to a file located in the central repository, a developer can get a working copy of the original file, which can then be modified and used to create a new version of the original file in the repository. While a developer is making changes to a file, CVC systems rely on a file locking mechanism that prevents other developers from making changes to that file until it is no longer being modified. This mechanism ensures that no one starts to work on a file which may soon become outdated. At the same time, while locked, the file cannot be modified by other designers, even if they intend to modify different parts of the program. In other words, file locking assumes that changes made to the same file by different developers will be incompatible, which may or may not be the case.

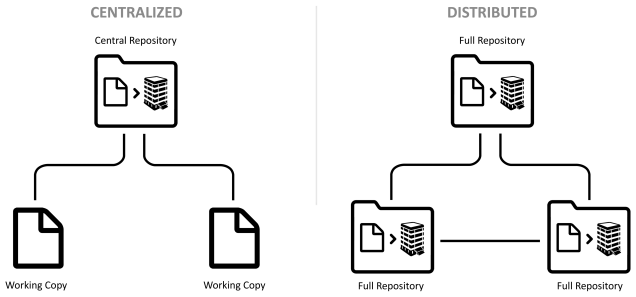

Figure 2. Centralized Version Control vs. Distributed Version Control.

On the other hand, DVC systems allow developers to work locally and independently in their own local repositories until they are ready to share their changes with other developers. This provides a safer environment for developers to experiment with the project without having to worry about the repository's integrity. Once a developer completes a set of changes, they can check the progress of the project to see if any other changes have been made before sharing their own changes with other developers. If so, unless changes are incompatible, they can be merged with other repositories with ease.

While software developers generally find CVC systems easier to learn and use, they usually prefer DVC systems due to (1) the ability to share the work only when it is complete and (2) a better record and management of software changes. Despite the steeper learning curve, DVC systems can offer higher productivity than CVC systems (Brindescu et al. 2014).

3. Versioning and Management Tools

Designing a building often involves a necessarily complex process. It requires collaboration among a large number of participants with different specialized skills, priorities, and objectives (Rosenman et al. 2007). Despite working towards a common goal, these participants typically work independently on different parts of the problem, each using the methods and tools that best fit their needs (Kvan 2000). This might lead to several individuals working concurrently with the same information and set of files, which can be a challenge. To mitigate this, tools have been developed to manage the different inputs produced by the multiple participants.

One of these systems is Autodesk's Vault, a management tool that tracks changes in the submitted files and maintains VC over these. Vault also supports collaborative development, since it allows several users to access and manipulate these files, which can also be locked, preventing changes from being made by several members at the same time. With Vault, users can also see the files' history, reverting them to previous versions if needed, or recovering deleted files.

Another system is ProjectBank, a collaborative engineering data repository by Bentley Systems. This system has similar revision capabilities as Vault but adopts an optimistic approach regarding access to the data in the repository, allowing more than one user to access the same data simultaneously. This strategy is used in order to improve parallel development, by assuming that there is a certain degree of coordination between team members to avoid conflicting changes. When a user is modifying a file, it is only possible to submit those changes if they are the only differences between the user's copy of the file and the server version. If there are other differences, this might lead to conflicts. To handle this issue, ProjectBank allows users to implement a *changeMerge* method to detect conflicting changes, which have to be solved with human intervention (Aish 2000).

While these tools facilitate collaborative work in design projects, they are tailored to deal with files generated by CAD or BIM tools and are not appropriate for algorithmic approaches, such as A-BIM, where the model is generated by an algorithm and a single change to this algorithm or to its inputs might generate a completely different model, making it very difficult to track the changes in it. To

solve this problem, the changes need to be tracked in the code itself. This means that VC has to be done in the code files instead of the model files that are repeatedly generated upon the execution of the code.

Code-related VC is well known in software development and there are already several tools that accomplish this task. In this paper, we claim that these tools are more appropriate to support design projects developed using algorithmic approaches and, in fact, some are already being used in architectural studios. One example is Subversion, a CVC system used by Gehry Tehnologies to support shared repositories for internationally distributed projects, such as the Foundation Louis Vuitton in Paris (Burry & Holzer 2005).

More recently, a DVC system became very popular: Git (Chacon & Straub 2014). Git encourages Branching and Merging to implement changes, allowing users to easily switch between different development branches, and making it easy to test solutions in different versions of the code. Git repositories are hosted by several services, such as GitHub, allowing for a wider collaboration, meaning that collaborators can download, test, review, and further develop the code. With the use of GitHub, several community projects have been developed for many areas, including architecture, e.g. DynamoDS [1] and Lyrebird [2].

4. Version Control for A-BIM

Despite the usefulness of CVC systems for small-scale projects, we consider that larger-scale A-BIM projects are better served by DVC systems since, when working collaboratively, design teams usually work independently on their own parts of the design of the building. DVC systems support this workflow by allowing developers to work locally and independently and only share their work when they are ready to do so. On the other hand, using CVC systems, submitted changes are immediately shared with other team members, regardless of whether the authors are ready to share them, or other developers are ready to accept them. Moreover, CVC systems have inflexible protocols (e.g., the file locking mechanism) and hierarchical structures, which can prove to be an obstacle for creative flexibility and concurrent development during early design stages (Burry & Holzer 2005).

In order to show how VC can be applied to A-BIM, the following sections describe a set of scenarios where it greatly benefits the development of architectural projects. We chose the Absolute World Towers, designed by MAD Architects, as a case study, which we modelled using an A-BIM approach with the Rosetta programming environment (Feist et al. 2016), using the Racket programming language. The model was generated in both Revit and ArchiCAD, the BIM applications supported by Rosetta. VC was supported by Git using the GitHub desktop client.

To start the project, a repository was created by one of the developers. The other developers cloned the repository, in order to have their own copy that could be developed independently from the original one. Every time a developer made changes to the project, they were recorded in the repository of that developer and were eventually shared with the others.

4.1. PURSUING DEVELOPMENT OPPORTUNITIES

When developing a design project, there might be several different ways to accomplish the same goals. For an approach like A-BIM, where the model is generated by algorithms, this might lead to the definition of different algorithms to test which one is preferable. Without a VC System, the designer either keeps different versions of the algorithm in the same file, of which only one is active at a time, or he uses different files, each with one version, managing the dependencies between files so that only one is actually being used. Either way, this is a cumbersome and error-prone task that might lead to bugs and loss of information.

Regarding the Absolute Towers' program, we initially wanted the code to be highly parameterized when modelling the elevators. We wanted the dimensions of the elevators and of the corresponding shaft to be parametrically calculated in relation to the available space, as well as in accordance with a set of regulations defined in the form of parametric constraints. However, we were not entirely sure that this approach would be feasible and so, to develop this idea, we created a branch to avoid disturbing the main line of development, with the expectation of later merging it with the main branch. In the end, this line of development was abandoned due to lack of time as well as limited payoff, since the development effort was too large for the time-frame available for the project. Nevertheless, if needed, Git still allows us to recover the abandoned branch to continue its development in the future.

4.2. FACILITATING CONCURRENT DEVELOPMENT

The previous scenario addressed a situation where there was only one designer developing the program. However, in a collaborative project, the program is developed by a team, and in order to have more control over the evolution of the project, it is important to keep track of the change history as well as each designer's contribution. Furthermore, when working concurrently, each designer is given their own copy of the project files that can be developed and tested independently from the others. When a task is complete, a designer notifies the others and they can incorporate the changes as they see fit.

Without VC, all changes have to be manually identified and merged, and are not recorded for future reference. Moreover, if some modifications are not identified, they can be overlooked when combining the code. On the other hand, DVC systems explicitly support this type of development scenario, making it easier to manage.

During the development of our case study, we divided modelling tasks among two designers. For this scenario, we will consider a task that required, on one hand, the creation of window frames for the tower, and, on the other, the introduction of guardrails on every floor. Using their own repositories, each of the developers worked in the corresponding part of the code. When the window frames were finished, the designer used the GitHub client to record the modifications in his repository. After that, a notification was issued to the other designer, asking him to accept the changes that were made. After this request was accepted, the code for the window frames was automatically merged, which means that it was included

in the other repository. An identical process was done when the guardrails were completed by the other designer, so that both designers had identical and complete versions of the code.

4.3. RESOLVING CONFLICTING CHANGES

When working concurrently on the same project, designers will often work independently on individually assigned tasks, which might result in two designers inadvertently modifying the same part of the program. Most of the times, these modifications can be easily combined to fit both designers needs, but sometimes modifications can be directly contradictory or too different to be combined. When these incompatible changes happen, one or both of the designers will have to find a way to peacefully solve the conflict. Like the former scenario, without VC, these changes have to be manually identified and merged, with the necessary measures to solve the conflict. However, VC systems have features to simplify conflict resolution, namely the identification of the conflicting lines of code.

As an example, during the development of the Absolute Towers, one update to the way slabs work interfered with code regarding the creation of floor levels. This occured because two developers changed the same lines of code for the floor levels differently and at the same time, which resulted in a conflict when trying to merge the changes back into the original repository. When using Git to merge the code, the conflict was identified and the application showed the parts of the code that were incompatible, allowing the user to choose which parts of the code should be kept to do the merge successfully.

5. Evaluation

By analysing the previously presented scenarios, it is possible to compare what designers can do with VC and how they could do the same thing without it. Table 1 shows this comparison for the three presented scenarios.

Table 1. Development of A-BIM projects with and without VC.

Task	Version Control Approach	Traditional Approach
Pursuing Development Opportunities	The designer's repository keeps every version of the code and additional development branches can be created.	The designer keeps different lines of development in separate files or as different fragments of code in the same file.
Concurrent Development	Each designer has a copy of the project that can be developed independently, and then safely merged.	All designers have to keep track of what was modified in order to manually combine changes or risk overwritting files.
Resolving Conflicts	The VC system shows where the conflict occurs and the designers decide which parts of the code should be kept.	The designers have to be able to identify the conflicting parts and manually resolve the conflict.

By analysing the first task in table 1, we note that, if designers wish to store possible lines of development for future projects, with the traditional approach, they have to save their code in different files. If the number of files is small, this is not a problematic task, and can be easily managed without VC. However, in more complex projects, if designers have to produce and maintain several dozen files,

this can quickly become very difficult and error-prone. Furthermore, using this approach, it is common to lose intermediate versions, which naturally disappear as the project evolves. With VC, each version is saved in the repository and it is possible to easily identify changes between versions.

Regarding the concurrent development of A-BIM projects, if two designers want to work simultaneously on the same file, either one waits for the other to finish his changes or they have to work on copies of the file and, afterwards, coordinate with each other to manually identify and combine the changes. On the other hand, with VC systems, users have full copies of the project that can be developed independently. Each time a designer wants to share his work, he requests a merge into the other designer's copy. If there are no conflicts, the changes are merged automatically, sparing a great amount of manual work that is unavoidable when working without VC.

Finally, when conflicts are introduced by designers changing the same files at the same time, VC systems can clearly identify the conflicting lines of code. Sometimes, the conflict cannot be automatically solved by the system without human intervention but, even in those cases, designers have clear information about what was changed and what is in conflict. This is helpful information for resolving the conflict that is not available without the use of VC.

5.1. SUGGESTED WORKFLOW AND LIMITATIONS

Taking into consideration the advantages of VC, we propose a workflow for the use of VC systems in algorithmic-based architectural projects. For this workflow, we recommend DVC systems, such as Git.

Considering the development of a building using an algorithmic approach, the first step would be to create a new repository dedicated to that project. Afterwards, each collaborator of the project must clone the repository containing the project files, in order to have access to it and develop their tasks independently. Each collaborator will have a set of tasks assigned to him, and will create a branch to develop the code for each task. During the development of the tasks, changes will be locally recorded, creating new versions, which allows developers to revert to older versions of the code if necessary. When a task is completed, a request to merge the code is issued by the developer of that task, so that others can, if they want to, incorporate his code in their repositories. If the changes do not cause conflicts, they are automatically merged by the VC system. Otherwise, the VC system identifies the conflicting code, and the conflict must be solved manually by the developers.

Although the proposed workflow is suited for the development of projects using textual programming languages, it has limitations regarding visual programming languages, such as Grasshopper. Since current VC systems are tailored for textual files, they can detect that the visual program was changed but cannot easily identify what was changed. To apply VC to visual programming languages, the system should be able to read the visual program and identify the elements that were added, deleted, or modified. To present those changes, these elements could be highlighted with a color scheme, e.g. green for added elements, red for deleted

ones, and yellow for modified ones. This is a promising line of research that we are considering for future work.

So far, we only discussed the tracking of changes in the code itself. However, it is interesting to be able to visualize those changes in the generated model. This requires *traceability*, i.e., a correlation between models and the code that generates them (Leitão et al. 2014), so that we can identify the changes in the code and trace them back to the model, which is something that VC systems are not capable of doing. We plan to take advantage of traceability to highlight in the model the changes between different versions of the program.

6. Conclusion

The design approaches used in Architecture have been changing throughout the years, becoming more sophisticated as technology evolves. More recently, AD approaches, such as A-BIM, have been adopted to develop more complex projects. Since these approaches rely on algorithms and code files that produce the model, instead of the model itself, the tools that were traditionally used for collaborative work with BIM and CAD applications are no longer appropriate. However, tools used in software development present a good alternative to work with algorithmic approaches.

VC systems allow software developers to, not only maintain versions of their programs, but also create parallel branches to explore alternative software features without interfering with someone else's work. These systems can also track changes and help developers merge their work, making it easier to have several developers working concurrently in the project files.

Due to the algorithmic nature of A-BIM, it is useful to take advantage of VC systems while developing projects with this type of approach. VC helps designers develop new versions of their algorithms, or retrieve old ones if necessary, as well as explore design alternatives in new branches of development.

We showed several scenarios, based on the development of a case study with A-BIM, where VC proved to be helpful to designers by making it easier to deal with several versions of code, parallel development, and even conflict management when two designers modify the same file at the same time.

By comparing what was done with VC with the alternative without these systems, we see that VC greatly reduces the amount of work designers have to do, by automating some tasks and making others easier to perform, such as tracking conflicting changes and helping designers solve them.

It should be noted that these strategies only work with textual programming languages. VC systems are designed to compare textual files in order to accurately detect differences between versions and classify those differences as conflicting or not. Due to their graphical nature, visual programming languages are not easily supported by VC systems, currently making textual programming languages preferable for this type of development.

We conclude that VC systems are useful tools for the development of projects that use algorithmic approaches with textual programming languages. In the future, we will continue to explore VC for collaborative architectural projects as well as

other software development practices that might be appropriate when developing projects with algorithmic approaches.

Acknowledgements

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

References

"Dynamo Source Code" : 2011. Available from Open Source Repository<https://github.com/DynamoDS/Dynamo> (accessed 23rd December 2016).

"Lyrebird Source Code" : 2014. Available from Open Source Repository<https://github.com/logant/Lyrebird> (accessed 23rd December 2016).

Aish, R.: 2000, Colaborative Design using Long Transactions and "Change Merge", *Proceedings of the 18th eCAADe Conference*, Weimar, 107-111.

Azhar, S., Brown, J. and Farooqui, R.: 2009, BIM-based Sustainability Analysis: An evaluation of building performance analysis software, *Proceedings of the 45th ASC Annual Conference*, 1-4.

Brindescu, C., Codoban, M., Shmarkatiuk, M. and Dig, D.: 2014, How do centralized and distributed version control systems impact software changes?, *Proceedings of the 36th International Conference on Software Engineering*, 322-333.

Burry, M.: 2011, *Scripting Cultures: Architectural Design and Programming*, John Wiley & Sons.

Burry, J. and Holzer, D.: 2009, Sharing Design Space: Remote Concurrent Shared Parametric Modeling, *Proceedings of the 27th eCAADe Conference*, Istanbul, 333-340.

Chacon, S. and Straub, B.: 2014, *Pro Git*, Apress.

Eastman, C., Teicholz, P., Sacks, R. and Liston, K.: 2008, *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers, and Contractors*, John Wiley & Sons, New Jersey.

Feist, S., Barreto, G., Ferreira, B. and Leitão, A.: 2016, Portable Generative Design for Building Information Modeling, *Proceedings of the 21st CAADRIA 2016 Conference*, Melbourne, 147-156.

Kolarevic, B.: 2003 (ed.), *Architecture in the Digital Age: Design and Manufacturing*, Spon Press, New York.

Kvan, T.: 2000, Collaborative Design: What is it?, *Automation in Construction*, **9**(4), 409-415.

Leitão, A., Lopes, J. and Santos, L.: 2014, Illustrated Programming, *Proceedings of the 34th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*, Los Angeles, 291-300.

Pottmann, H., Eigensatz, M., Vaxman, A. and Wallner, J.: 2015, Architectural Geometry, *Computers & Graphics*, **47**, 145-164.

Rosenman, M., Smith, G., Maher, M., Ding, L. and Marchant, D.: 2007, Design World: Multidisciplinary collaborative design in virtual environments, *Automation in Construction*, **16**(1), 37-44.

Santos, L., Lopes, L. and Leitão, A.: 2012, Collaborative Digital Design: When the Architect Meets the Software Engineer, *Proceedings of the 30th eCAADe Conference*, Prague, 87-96.