Automated Code Compliance Checking

A computational workflow for verifying model, parameter and regulatory compliance

Luka Jovanovic[1,2], Leo Meng[2], Ben Doherty[1], Nicole Gardner[1], M. Hank Haeusler[1], K. Daniel Yu[1]
[1]University of New South Wales / Computational Design / [2]HDR Architects Sydney
[1]{l.jovanovic; b.doherty; n.gardner; m.haeusler; d.yu }@unsw.edu.au [2]{luka.jovanovic; lin.meng@hdrinc.com

Building Code Compliance Checking (CCC) is a necessary part of the design process in the Architecture, Engineering and Construction (AEC) industry that is typically time consuming. Existing Automated Code Compliance Checking (ACCC) solutions can be considered as 'Blackbox' and are often location specific. Using action research methodology this research develops the design of the computational workflow led by an 'informed research participant' – an industry partner – who has knowledge of the problem. All to create a visible and customizable script to evaluate model compliance as well as CCC results. The research outcomes demonstrate a promising compliance checking computational workflow method that could be applied to a range of building codes and standards and contributes to building the AEC industry's confidence in workflow automation to drive more productive and sustainable ways of working. This investigation, its proposed hypothesis, methodology, implications, significance, and evaluation are presented in the paper.

Keywords: *Automated Compliance Checking, Regulatory Compliance checking, BIM family, Grasshopper (GH), Revit.*

INTRODUCTION

There are a variety of codes in the Australian Standards, as well as other sets of standards which help create a safer and more accessible building. Building Code Compliance Checking (CCC) is typically a time-consuming process, and if compliance errors are found during the construction phase, it can lead to delays and incur significant costs for rectification work. To improve the speed and accuracy of CCC, Automated Code Compliance Checking (ACCC) solutions such as Solibri Model Checker (SMC) and Upcodes have been developed. Revit files must be exported as an IFC file for SMC and is also considered 'black-box'. 'Black-box' workflows provide a solution which is not clarified to the user, and often contains minimal ways to customise the process. A Revit plugin known as *'Upcodes'* provides a degree of clarity and customisability, though it only allows a user to select from various US standards, so is lacks use in other contexts. This workflow tool is targeted towards the compliance checking of doors in accordance with Australian Standards (AS1428.1:2021). Compliance checking is also often limited to checking codes and does not check whether the families within a project are made under the correct modelling and naming conventions. This research adopts an Action Research methodology. This involves an iterative process of design in collaboration with an industry partner that responds to an identified real-world problem in the AEC

industry. In this way, the design of the computational workflow is led by an 'informed research participant' who has knowledge of the problem. Initial efforts begin in a default Revit project in order to understand the process of extracting Revit parameters through Rhino.Inside.Revit's (RIR) Grasshopper (GH). Next, it is prototyped using HDR Architects door family company models, which include robust modelling standards for Revit door family parameters. A workflow is created to evaluate model compliance as well as CCC results. As the workflow is easily visible and customizable in GH it can be considered a 'white box' solution. The GH script ensures that Revit family parameters, name, and model dimensions all match. The workflow automatically checks that Revit family names and models dimensions correspond and flags non-compliant results by highlighting the models and through an error message. The workflow checks the model file for dimensional compliance including critical clearances and widths as per AS 1428.1:2021. Unlike many other current industry standard approaches, this research develops a workflow that also checks model integrity. As well as this, it automatically uses vectors to detect the approach sides of doors based on surroundings, giving it a unique 'awareness'. The research outcomes demonstrate a promising compliance checking computational workflow method that could be applied to a range of building codes and Standards in Australia as well as elsewhere. It creates a solution which is faster and accurate to the standards, ensuring that costs due to mistakes do not occur. Developing 'white box' solutions such as this is critical to building the AEC industry's confidence in workflow automation to drive productive and sustainable ways of working.

RESEARCH AIMS AND QUESTION

The aim of this research project is to create a workflow which is able to check BIM door families in a Revit project in order to flag doors which do not follow the AS 1428.1:2021 (Design for Access and mobility) Section 10 (Doorways, Doors, and circulation space at doorways). In order to ensure an accurate checking process is being undertaken, the door models will also be checked for model compliance to ensure that the model's name, model parameters and model dimensions match. This error will also be flagged in a visually differing way to depict a lack of model compliance. With this in mind the research asks the following question:

How can a computational design workflow be developed to automate the extraction of parameter data from BIM families for compliance checking against codifiable aspects of one, exemplarily used standard - the Australian standards 1428.1:2021, while ensuring model compliance?

BACKGROUND RESEARCH & LITERATURE REVIEW

CCC of Building BIM models has historically been *"…time-consuming for architects and construction engineers"* (Bus et al. 2019, p.6) due to the manual process. This manual process *is "…based primarily on construction plans (two-dimensional technical drawings) and additional textual documents"* (Preidel et al. 2016, p.403) and has room for improvement due to its *"…laborious, cumbersome and error-prone"* (Preidel et al. 2016, p.403) processes. This importance can be highlighted through the suggestion that *"…40% of building defects in Norway can be related to mistakes or omissions in the design process"* (Hjelseth 2009, p.2). CCC is a system which can be sped up, optimized and automated using technology. The commonly perceived obstacle of current State of the Art (SOTA) solutions for Automated Code Compliance Checking (ACCC) checking of BIM models, is that they are 'black-box' tools. The CCC solution proposed by such a black-box program, is a *"…code checking functionality within rule templates, which are based on hard-coded rules"* (Preidel et al. 2016, p.404). The problem which arises with such methods to model checking is *"…non-transparency and inflexibility to regulatory changes"* (Dimyadi et al. 2013). This inflexibility creates a need for transparency on how the checking happens, as well as customizability. 'White-box'

methods, on the other hand, aim to overcome the lack of transparency and customization to compliance checking through the *"…mapping of objects and attributes…"* (Amor et al. 2021) which are *"…specified and predefined"* (Ibid., p.4) by people. This creates a baseline of an ideal ACCC system to aim for when creating a design workflow for checking the model and code compliance. It could help view the processes taking place, editability, as well as make local model checking standards easily customizable.

Translation of Rules, Standards and BIM data.

Machine Readability. The current problem with current standards is that they don't *"…allow for a direct translation into machine-readable representation"* (Preidel et al. 2018). A problem can also arise when rules or standards are misinterpreted, causing *"…errors during the extraction of the rules"* (Malsane et al. 2015, p.54) which can be avoided through domain expert translations. When translating standards into a computer-readable form, it is advantageous to make them human-readable to create *"…a practical and efficient approach."* (Preidel et al. 2016, p.404). In the case of the Australian Standards 1428.1:2021, this could be quite easily translated since many rules are diagrammatic or numerical. The translation process is most advantageous when the ACCC workflow can *"…check and satisfy many rules within domain specific problems or best practice"* (Solihin et al. 2015). This ensures that the workflow can use all the model information needed to perform each check.

Required Level of Detail. A challenge of ACCC is that many models do *"…not typically include the detailed level of information required for fully automated rule checking"* (Malsane et al. 2015, p.52). Such can be seen through a case study, where BIM models *"…with a level of details more than LOD 300 for efficient automated-checking process."* (Narayanswamy et al.2019, p.1047). This indicates that under ideal circumstances, a BIM model is at a high level of detail to allow for CCC calculations. The exact needed parameters for CCC may be found through further studies.

Advantages and Workflow of Automation. The SOTA model checker Solibri has a *"…rule engine that can work on multiple models"* (Nawari 2019, p.3). The reusability of such rule engines is advantageous, since it must only be created once, and then have models and standards inputted in to gain compliance results. When creating a ACCC workflow, it is important to be aware of the time saved due to reusability. Automation also leads to *"objective results"* (Dieckmann et al. 2014), which are *"considerably more precise"* (Ibid., 2014). A further benefit is that processes which are *"too tedious"* (Ibid., 2014) become achievable. Dieckmann and Russel also claim that such a system also helps users learn through automated feedback. *'KBIM'* is a programming language which includes *"…information extraction of the rules based on objects, properties, and high-level methods stored in SQL database tables, and KBimCode…"* (Sydora et al. 2020, p.3). According to Sydora et all. (Ibid.) explanation of KBIM Code, the workflow functions through inputs which are taken using BIM data, and results are extracted.

Visual Code

White-Box. A main advantage of visual coding is its white-box nature, since *"…the user is able to understand and inspect every single processing step…"* (Preidel et al. 2015, p.406). Visual Code Checking Languages are *"…extendable as the project matures"* (Sydora et al. 2020, p.4), granting the user the ability to *"…build around other predefined nodes"* (Ibid.). An advantage of visual codes is that they're *"…processed with two instead of only one hemisphere of the human brain in parallel"* (Priedel et al. 2016, p.406). This points towards visual white-box methods as an ideal situation which is easier to interpret and use. Visual advantage is also evident in Revit views, which are described as an *"educational tool to enhance understanding of the code and the building"* (Clayton 2013, p.7). Although Revit views are not a form of coding, it further depicts that

visualization eases the understanding more complex systems.

KBIM. Kbim has created a *"…VPL version of KBimCode to improve ease of use"* (Sydora et al. 2020, p.4). A visual programming language has been implemented into Kbim, though there are *"…no studies to verify this does improve usability"* (Ibid.). Although there hasn't been proof that visual programming language adds to the ease of KBIM use, explorations of this direction suggests that a visual programming language may help create a white-box, user-friendly ACCC workflow.

Optimo and Visual Scripting. This is evident in Optimo for Dynamo for example, which grants users the *"… option to optimize multiple objective functions with respect to multiple parameters among the rich data stored in BIM"* (Asl et al. 2015, p.681). Scripting on Revit allows for its Revit API to be harnessed through an *"…add-on software application…"* (Narayanswamy et al. 2019, p.1046). Traditional methods to pull information from an API *"…requires a high level of programming knowledge, even for the simplest checks"* (Sydora et al. 2020, p.3). This ability of visual scripting tools to access the Revit API points towards Dynamo or GH, to be valid options for ACCC, with the ability to pull information directly from Revit's API without a high barrier of entry, such as a traditional programming language.

State of the Art (SOTA). SOTA model checkers will make *"…assumptions and algorithms to find or use information that 'normally' fits"* (Hjelseth 2009, p.3) when the relevant information is not included. To create a white-box workflow, it may be important to list when information is absent in a step of the workflow, rather than silently replace information.

To conclude. Manual methods of CCC are time-consuming processes, and often result in and carry compound errors due to their manual nature. Current SOTA CCC tools are often black box, containing little to no customization. A goal of the automated checking process is *"…generating algorithms for the data exchanges between the components of the framework to execute the virtual review process…"* (Nawari 2019, p.5). Results appear to be promising in the scope of digital programming languages such GH, Dynamo, Optimo and Visual Code Checking Language. Inabilities of current tools are constantly being updated, so *"…limitations that are going to be addressed in the future releases…"* (Asl et al. 2015, p.681) such as time optimization for Optimo or extra clarity and customizability in SOTA CCC workflows. There is potential in visual scripting, which provides user-friendly automation, connection to the Revit API, and ability to automatically check based on predefined rules. Due to the integration of expert knowledge, *"fields where you are not expert can be checked."* (Hjelseth 2009, p.2). Further developments may address more Standards, where an accurate and customizable white-box meth- od may be used to address ACCC of a particular element, such as doors. Further studies may additionally unveil which LOD is needed for finding which parameters are required for a compliance and model check.

METHODOLOGY

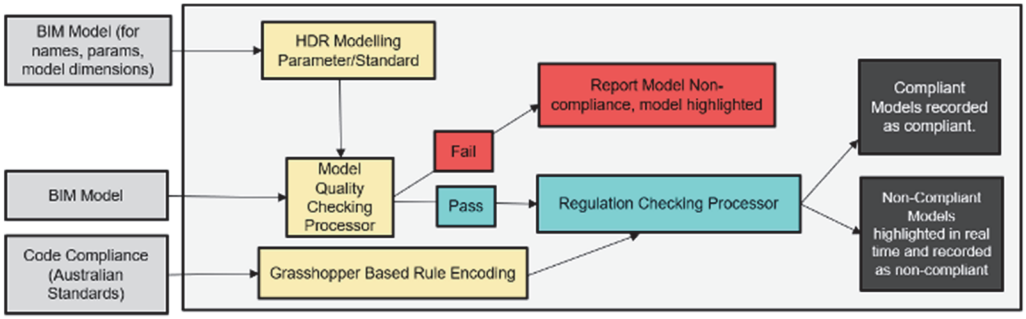
"Action research is operationalized by constant cycles of planning, acting, observing and reflecting" (Hearn et al. 2005, p.2). Although said cycles exist, *"… the observer stance leads to a disjunction between theory and action."* (Ibid, p.4). This is as there is a need to explore in the action phase and branch out from the research at hand. A predominate product nowadays is *"… digital, weightless and intangible, which makes them universal and more and more pervasive and ubiquitous"* (Ibid., p.4). Such would be applicable to Automated Code Compliance Checking (ACCC), which would be adopted in a ubiquitous manner. A question by the author *"… how can we expect to find a solution to a problem if we do not know what we are looking for"* (Ibid., p.4) gives a reason for action in research. This is later depicted in three main ways: (1) *"Active participation: the people who should benefit from the research participate in defining the aims and direction of the research and in interpreting and drawing conclusions..."*; (2) *"Action-based methods: the activities and experiences of participants generate knowledge alongside, or in combination with, more*

formal methods", (3) *"…Action-generating research can be a combination of general, wide-ranging, background research and very specific focused research."* (Hearn et al. 2005, p.8). Such are methods of finding a result through action research. The 'end product' must be planned at the start, so a qualitative research method may be used as *"Qualitative methods tend to be best for generating theory and quantitative methods tend to be best for testing theory (Howard, 2002: 569)"* (Hearn et al. 2005, p.8). In the case of ACCC, it's clear that a model checker must be qualitative, as new iterations are created and observations are created, and industry partner feedback is considered. Quantities may also be useful to calculate accuracy in comparison to Australian standards and clash detection box measurements from the industry partner HDR. *"It is necessary to vigilantly maintain a clear distinction between action which advances open inquiry and instrumental action for its own sake"* (Hearn et al. 2005, p.15). This makes clear, that in the project timeline, action taken must bring forth more questions, as this allows for further discoveries. Such includes exploring current methods of interacting with BIM families, such as visual scripting in order to extract parameters and insert them into a workflow. Collecting data on the relevant Australian standards, as well as gathering data on the functions of current industry standard model checkers is also important, in order to understand what is yet to be achieved. Feedback collection is another important method; feedback is collected from our industry partner to verify whether parts of the script work as intended, or to gain further suggestions. Using this methodology, action research may be undertaken throughout the process of the project timeline, to produce a non-tangible goal in the form of a visual script.

CASE STUDY

This case study explores the iterative development of an ACCC workflow to check doors against AS1428.1:2021. The workflow is made to test doors directly in Revit through GH which is accessed via Rhino.Inside.Revit. Using Grasshopper (GH) allows for a white-box workflow to be created where the set of processes is clear and customizable. Through starting with a set of goals agreed upon with the industry partner, an ACCC is iterated upon as new features are constantly added. As well as new features, feedback is also considered to improve the capabilities of the workflow with each iteration. The geometry generated by the workflow is also compared and contrasted with the professional HDR door families and their clash detection boxes. Standards form another area for checking accuracy of the final workflow (see Figure 1).

Figure 1
High level workflow



Initial Iterative Exploration

Initial Door Family Extraction (Rhino 6). Initial efforts were made in Rhino by exporting the Revit project as dwg. Door leaves could be extracted; however, parameters lacked consistency among diff. door families as some were reported as null.

Initial Door Family Extraction in Rhino.Inside.Revit (Rhino 7). The ACCC was at then chosen to occur within Revit through RIR due to the software being able to work within Revit.

This extraction of the door leaves was based on the geometric rules, and thus only one door was selected from each door; this can be viewed in Figure 2. Door leaf parameters were found through the extraction and merging on instance parameters and type parameters.

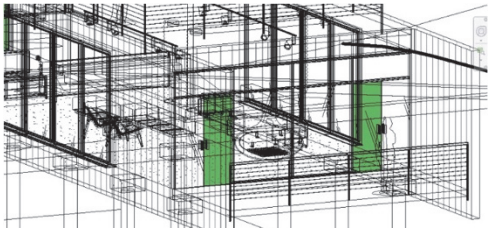


Figure 2
Initial Rhino.Inside.Revit door leaf extraction

Door Family Extraction on Robust Families. A file was supplied by industry partner including their robust door families. This would serve as a practical real-world example for door families. Extraction of door width, door height and door thickness were then findable through their respective type parameters. A user would have to input the names of the type parameters for each situation needed for leaf thickness, width, and height. The doorway depth is also calculatable; for surface mounted or sliding doors, this depth is the wall thickness; for the centered pocket sliding doors and pivot doors, this measurement is calculated by: (Wall Thickness- Door Leaf Thickness)/2. Swing doors are set to one side of a wall, so its dimensions are Wall Thickness - Door Leaf Thickness.

Initial Efforts in Measuring Standards

To measure the distance away from the door 'L', a line was extended from the sides of the door and then trimmed when it hits a wall. It can depict the distance away, as is required by the Australian standards, though this would be an inconsistent method for detecting objects within the area of a realistic clash detection area. To generate a line between walls, both walls also had to be manually selected, which would add a repetitive and unnecessary manual task to the workflow. During this stage of the project, clean door leaves without extra geometry was extractable, and was used to find the extents of the door. These extents were found and plotted in the middle of the wall through finding the closest points of the door extents to the respective wall center line. Door type names were also converted from their HDR door types to Australian Standard door types through a GH script. This was for name simplification of further processes, however, is not necessary for further processes. A python script was also implemented to automatically define what the limit for Distance 'L' would be based on the Door Type, Clearance Distance, whether the hinge is flipped, whether the door is flipped, the approach side (which was at that stage placeholder data), as well as the potential Australian standard values depending on the door. The script uses the door type check to figure out the maximum distance away for the swing direction of the door, as well as the non-swing direction of the door. Based on whether the door is flipped, and if the hinge is flipped, the direction of approach can then be determined, to be used for a check. Error messages were given if the door type was.

FINALISED ITERATION – REGULATORY AND CODE COMPLIANCE

In the final iteration, the script was cleaned, and every process was labelled. At the start of the script, after entering the Australian standards, the user would flow from left to right to be able to select the geometry needed through RIR's built-in category node. The script is set to selecting 'Windows', 'Walls'

and 'Doors' parameters, but allows for changes if a company using the script has different naming conventions. The parameters can then be selected underneath.

This allows the script to call for the correct parameters when needed throughout future processes. A user also can tweak the default values of the limit of the depth at a doorway, which is set to 300mm by default to match AS1428.1:2021. The threshold for the distance which defines an approach side is also customisable. This is where door indexes not to include for clash detection can also be defined, as well as the geometry to use for vector detection, and geometry to use for clash detection. Geometry used in clash detection and vector detection should have all lines removed before being placed in this process or an error may occur in clash detection. All of these settings and customisation are also at the start of the script after the Standards inputs. A model checker was added to ensure that the model is not erroneous and that all end results are reliable. It checks the dimensions of the geometry against the dimensions of the parameter; the dimensions of the geometry against the dimensions in the name; the dimensions of the parameter against the dimensions in the name. These will give a compliant or non-compliant result in panels in a part of the GH script. The uncompliant doors will also be highlighted in the project. In the Rhino view, the index of each door is also shown, so it makes it easier to match the door to its respective compliant or non-compliant result based on the index. The clean door leaves without extra geometry are extracted to allow for dimensions and simple geometry to be easily extracted. The cleaned door leaves are simple boxes and taken by extracting the leaves and creating a bounding box. Vectors were utilised in order to find the approach side. There is a front side approach for the swing and non-swing side of the door. The hinge and latch directions are also calculated through vectors. The vector clashes with an object, the respective side is not counted as an approach direction. The vector distance is 1600mm long, which is an appropriate distance for detecting whether something is an approach distance according to the industry partner. Vector visualisation and length measuring process is done through the DeCodingSpaces GH plugin.

Due to the line-based method for finding distance 'L' could only find if something in the way of the line, rather in a volume, the line-based method was replaced with a clash detection box, which is more accurate. This part of the script allows for default values or custom values to be chosen and switched between through a number slider. The chosen values are converted into a large list of lists depending on the approach side for each door. To check whether a sliding door is a surface mounted sliding door or not, the door is moved down in the Z axis by 200mm and checked for whether it clashes with the wall center line. If it clashes, it must not be a surface mounted sliding door; and then gives an output for each sliding door, 'Centred' or 'Surface_Mounted'. There is a new version of the python since the Distance L Detector.

The script is capable of outputting the needed Australian standard distance for each needed situation. It considers the door type, clearance distance, swing side approaches, non-swing side approaches, AS1428.1:2021 clearance values for each door type (Swing, Pivot Hinge, Sliding), error messages for doors not included in AS1428.1:2021 (Folding, Rolling), whether the door is a single or double door, Leaf thickness, and the sliding door type (Surface mounted or not). The measurements for the Front vectors are flipped if the door is flipped according to the Revit API through a python script. The hinge and latch side are measurements are flipped if the hinge side is flipped according to the Revit API. If the door type is a double door, only hinge side approach measurements are outputted, since there are always hinges on the outside of double doors. The outputted list values are then deconstructed into hinge side distance, latch side distance, and the distance away from the door for every single door. The front surface is offset to line up with the wall and then 1mm is added to the distance to allow for the host wall to not clash with

the host wall. Then the hinge and latch distance sides are offset by their previously determined offset distances in the last python script. The clash detection boxes are then created by using the midpoints of each surface and extruding that surface along that curve to create the solid box in front of the door. The same method is used for both swing and non-swing sides of the door, with their respective measurements being used.

The final clash detection boxes for the front and back are then used to detect for clashes against walls, doors, windows, furniture, or any other object which would be defined by a user. If clashes occur, then the clash detection box will be visible in the Revit as well as the Rhino viewport. Along with this, results of clash detection will be displayed in a panel at the end of the GH script. This allows for ease of use and multiple ways of being able to recognize if a clash has occurred. Additionally, the door index and side of the clash are also written in the panel, giving extra context to the user. There is also an option placed at the start of the script which allows the user to select doors to exclude from the clash detection test. When testing the clash detection boxes against the manual clash detection boxes in the industry partner's door model, a few differences were found between the boxes. The final clash detection boxes matched the values in AS1428.1 for maximum width and depth and didn't account for the front and side approaches having a differing maximum length away. This may be resolved in future exploration through creating two boxes with their respective extents and combining them through a solid union. The clash boxes from the industry partner also sometimes differed in maximum length, however it seems this may be due to extra length as padding, or another standard not measured in this research.

The final script was tested on the default Revit project with door families from the industry partner, the generated clash detection boxes consistently worked in cases they were supposed to. The final results generated accurately represented the clash detection boxes in areas where compliance where AS1428.1:2021 failed. It also accurately highlighted door leaves in cases where the door model was incorrect, and parameters did not match. This final script was tested in the Revit default project with the doors being replaced with families from the industry partner. This can additionally be viewed in the Rhino viewport to see door indexes, and for more control over surrounding object visibility through GH.

DISCUSSION

The research question *'How can a computational design workflow be developed to automate the extraction of parameter data from BIM families for compliance checking against codifiable aspects of the Australian standards 1428.1:2021 while ensuring model compliance?'* has been successfully answered through this research. The research successfully resulted in a workflow which achieved the aims. Doors which do not follow AS1428.1:2021 are flagged and model compliance is ensured through testing whether the name, parameters and the door leaf dimensions match. The ACCC workflow was made using GH through Rhino.Inside.Revit (RIR). A user may enter parameter names, thresholds, and Australian Standard Values if they wish to change AS1428.1 default values. Door model compliance ensures the model's name, model dimensions and model parameters match. Approach side is calculated through vectors created through DeCodingSpaces' Isovist3D, as well as the Revit API to find the front of a door, and which side is the hinge or latch. The approach side and [LOCATION] Standard values are then used to automatically create clash detection boxes. The ACCC workflow took approximately 30 seconds to load upon start-up as well as when doors are moved, or values are changed to the initial calculations. It can check HDR doors or any other doors if needed parameters are entered. When a model compliance check fails, it highlights the door, and a text box gives a door index and a reason.

When a code compliance check fails, the clash detection box is highlighted, and the door index is given, as well as the number of clashes. This is viewable in the Rhino or Revit viewports.

Additionally, clashes and uncompliant models may be highlighted as green instead of red through highlighting the clash results in GH. The workflow created is far less time consuming than manual CCC. It takes around 30 seconds to check 11 doors, and with further optimisation may work smoother. The clash detection boxes are not always completely accurate, as they may contain slightly more volume than they should. This may be improved by combining 2 boxes, one for the left and right approaches, and one for the extent of the distance away. The current User Interface (UI) is a heavily labelled and colour coded GH script, which is simple to understand, however but a better UI, or the GH player function could create even simpler user experience. More display options may allow for indexes to also be viewable in the Revit viewport, and for erroneous areas to be highlighted in customisable colours in Revit. Further customisation options such as a manually choosing approach side could be implemented in future works. Similar processes may help automate the remaining numerical Australian Standards. The addition of Automated Model Checking and Automated Code Compliance Checking to the AEC industry allows for improvements which help improve the workflow in terms of speed, error reduction, but also tend to the inclusive needs of people with mobility issues.

CONCLUSION

Compliance checking is a task which ensures that a building is safe, accessible, and inclusive. Current industry standard software often doesn't allow for 'white box' workflows, meaning that customizability suffers. The user is also unable to see what happens in each process and thus is also unable to verify the processes occurring are correct. This research project resulted in a GH and Revit workflow, which uses RIR to run Rhino 7's GH. This workflow was achieved through action research methodology. The resulting workflow allowed for processes to be easily viewed and customized by a user. The customizability factor means that a user has the freedom to alter the measurements to match the standard in any global context, or if standards are to change, given the same measurements are being taken. Through iterative design, the workflow achieved was able to automate the regulatory compliance checking of models to ensure they are modelled correctly, as well as automate the process of compliance checking against rules in the Australian standards. The workflow was able to check doors geometry-based rules in design for access and mobility standards in AS1428.1:2021 for the doors in a building. The contribution of this research is this Automated Code Compliance Checking workflow, which is 'aware' of geometric relationships; and with further developments into processing time and UI, as well view customizability, a similar workflow could be adapted to work for other sets of standards. Such a workflow could prove useful to the AEC industry to save the time and money through removing manual checking, while accurately completing results, using measurements which are confirmed by or altered by the user.

REFERENCES

Al-Hussein, M, Liu, H, Narayanaswamy, H (2019). BIM-based Automated Design Checking for Building Permit in the Light-Frame Building Industry, 36th International Symposium on Automation and Robotics in Construction (ISARC 2019), avail. at: www.researchgate.net, (Access 29.9.21).

Amor, R, Dimyadi, J (2013). Automated Building Code Compliance Checking – Where is it at? Avail. at: www.researchgate.net, (Access 20.9.21).

Amor, R, Dimyadi, J (2021). The promise of automated compliance checking, Developments in the Built Environment, vol.5, avail. at: www.sciencedirect.com, (Access 20 September 2021).

Asl, M.R, Stoupine, A, Zarrinmehr, S, Yan, W (2015). Optimo: A BIM-based Multi-Objective Optimization Tool Utilizing Visual Programming for High Performance Building Design, Real

Time, avail. at: www.researchgate.net, (Access 20.9.21).
Borrmann, A, Bus, N, Fies, B, Priedel, C (2018) Pre-Processing IFC Building Models for Code Compliance Checking based on Visual Programming, avail. at: www.publications.
cms.bgu.tum.de, (Access 20.9.21).
Borrman, A, Preidel, C (2016). Towards Code Compliance Checking on The Basis of a Visual Programming Language, Journal of Information Technology in Construction, avail. at: www.itcon.org, (Access 29.9.21).
Borrmann, A, Priedel, C (2015). Automated Code Compliance Checking Based on a Visual Language and Building Information Modeling, avail. at: www.researchgate.net, (Access 20.9.21).
Bus, N, Fahad, M, Picinbono, G, Roxin, A (2019). Towards French Smart Building Code: Compliance Checking Based on Semantic Rules Linked Data for Architecture and Construction (LDAC'2018), avail. at: www.arvix.org, (Access 20.9.21).
Clayton, M.J (2013). Automated Plan Review for Building Code Compliance Using BIM, avail. at: www.researchgate.net, (Access 30.10.21).
Dieckmann, A, Russel, P (2014). The Truth Is In The Model Utilizing Model Checking to Rate Learning Success in BIM Software Courses, avail. at: www.papers.cumincad.org, (Access 20.11.21).
Eastman, C, Solihin, W (2015). Classification of rules for automated BIM rule checking development, avail. at: www.sciencedirect .com, (Access 20.9.21).
Foth, M, Hearn, G.N (2005). Action Research in the Design of New Media and ICT Systems, Kwansah- Aidoo, Kwamena, Eds. Topical Issues in Communications and Media Research, pp. 79-94
Greenwood, D, Lockley, S, Love, P.E.D, Malsane, S, Matthews, J (2015). Development of an object model for automated compliance checking, Automation in Construction, avail. at: www.sciencedirect .com, (Access 20.9.21).
Hjelseth, E 2009, Foundation for Development of Computable Rules, avail. at: www.itc.scix.net, (Access 20.9.21).
Nawari, N 2019, A Generalized Adaptive Framework (GAF) for Automating Code Compliance Checking, Buildings, avail. at: www.researcgate.net, (Access 20.9.21).
Stroulia, E, Sydora, C 2020, Rule-based compliance checking and generative design for building interiors using BIM, avail. at: www.sciencedirect .com, (Access 20.9.21).