Compliance checking on building models with the Gherkin language and Continuous Integration

Moult, Dion; Krijnen, T.F.

DOI
10.14279/depositonce-9977

Publication date
2020

Document Version
Final published version

Published in
Proceedings of the EG-ICE 2020 Workshop on Intelligent Computing in Engineering

Compliance checking on building models with the Gherkin language and Continuous Integration

Dion Moult, Thomas Krijnen [1]
[1] Delft University of Technology
t.f.krijnen@tudelft.nl

Abstract. In this paper we document our approach on applying Behaviour-Driven Development (BDD) and Continuous Integration (CI) from the software industry to the construction sector. We have provided a freely available open software toolset for the application of rules in the Gherkin syntax to an IFC building model. A prominent aspect of BDD and contrary to mvdXML, the formalization of rules in plain-test human-readable scenarios provides a basis for collaborative formalization of rules among stakeholders. At the same time our approach includes imperative program code that is fully extensible to incorporate for example external data sources and geometrical reasoning. Runnings test on every model revision (the CI concept) as opposed to, for example, upon model delivery ensures a proactive approach to compliance. Reusing existing open source frameworks allowed us to build a comprehensive solution for continuous and automated model checking, visualization and reporting in several hundred lines of program code.

1. Introduction

The Industry Foundation Classes (IFC) cover a wide variety of subdomains in the construction industry and offer a considerable degree of freedom to modellers and implementers. Model View Definitions (MVD) can be used to formulate additional requirements on top of the rules and specifications defined in the IFC schema. The use of MVD is intended to specify requirements for certification purposes within the standardization organisation as well as requirements that are specific to individual projects or organizations. In short, it is assumed that if a model is delivered that complies with an MVD, it is fit for the purpose of a defined user.

Various MVDs are published as an international standard, four official standards for the IFC2X3 schema, and six standards for the IFC4 schema, four of which are still in draft form. Due to the limited publication, the current selection of MVDs do not account for the full variety of subdomains in the construction industry. Even within a published MVD standard, a recipient will often have different requirements compared to what is in the official standard. In addition, only one of these MVDs has an mvdXML available for distribution. Without a distributed mvdXML, it is complex to automatically validate deliverables against MVD requirements. Should the user wish to modify an MVD to their needs, they are required to modify the underlying mvdXML, or write their own.

The language to define MVD, mvdXML, has four main issues (a) the semantics are not precisely defined, (b) there is a variation in schemas and versions supported by tools, as such there is very limited interoperability, (c) the process of authoring mvdXML files is complex and requires extensive knowledge of the IFC schema, and (d) since the language is invented from scratch it's a large implementation effort for a vendor to implement support for mvdXML. Currently, the only free or open tool to compose mvdXML definitions is IfcDoc, which is not cross-platform. If an author wants to create their own mvdXML, this becomes problematic.

The published MVDs are primarily concerned with whether IFC entities are within the scope of a domain. Although possible within the mvdXML schema (using TemplateRules and in newer revisions of the standard: Constraints), the published MVDs do not help match attribute or parameter values to project-specific information requirements, which is often the priority to

a recipient of a BIM model. Therefore, the published MVDs are not sufficient to establish whether a model produced with an MVD is fit for purpose.

Most of the published MVDs include a large portion of the IFC schema. As MVDs are also used for certification purposes, this requires vendors seeking certification to be able to support the import or export of a large portion of the schema. This complicates the certification process, and discourages the development of smaller, Unix-like, targeted tools that manipulate a specific subset of the IFC model, such as geometry, or asset management metadata. This requirement for a monolithic implementation acts as a barrier against smaller vendors attempting to enter the market.

Currently, buildingSMART, the standardisation organization behind IFC and mvdXML is contemplating to embrace more industry-standard tools1. This paper provides an exploration of the application of an industry standard tool in validation and testing of software. This is the Gherkin language as implemented in software tools (such as Cucumber, JBehave, Lettuce, Behat, and Behave) for Behaviour- or Test-Driven Development (BDD/TDD). BDD principles include to obtain a shared understanding of how the application should behave by means of conversation and concrete examples. The authors believe that this practice is just as relevant to building codes as it is to software.

2. State of the art

2.1 Schema languages

The IFC schema is published in the EXPRESS modelling language, with programmatically derived copies in XSD and OWL. The need for model schema subsets has been described in the introduction. Both the EXPRESS and the XML families of standards include parts to define model transformations and views. EXPRESS-X (Baily et al. 1996) defines SCHEMA_MAP and VIEW constructs and reuses the procedural rule language in EXPRESS to further qualify or derive attribute types. Within the IFC community there are infrastructure domain extensions developed in UML. That family of standards include QVT for transformations and OCL for constraint languages. The Semantic Web standards include SPARQL for querying and transformation using SPARQL CONSTRUCT and SWRL for rules and SHACL for constraints.

As noted, the buildingSMART organisation put forward mvdXML[2] for this purpose and implementations of mvdXML exist, such based on SPARQL SPIN in Zhang et al. (2015)by which portability of rule execution increases, but at the same time a dependency on linked data technology creates additional indirection due to the mapping to OWL, in particular for lists (Pauwels et al. 2017). An effort to simplify the MVD definition language starting from mvdXML semantics is given in (Liu et al. 2019).

In this context it's necessary to appreciate the distinction between declarative and imperative languages. Where the imperative focuses on how something is to be executed the declarative focuses on a specification of the outcomes. Examples of declarative programming languages include the query and rule languages above, imperative languages are most of the contemporary programming languages such as C++ and Python. Pichler et al. (2011) concluded in their tests in the field of business process modelling that the imperative modelling approach was more comprehensible (although notes that this observation may be skewed by familiarity of the subjects). Intuitively one might argue that incremental changes are easier to apply to imperative

[1] This is seen in initiatives such as https://github.com/buildingSMART/NextGen-IFC
[2] https://standards.buildingsmart.org/MVD/RELEASE/mvdXML/v1-1/mvdXML_V1-1-Final.pdf

statements, but declarative approaches might have a higher degree of composability, the ability to combine statements is less hindered by imperative considerations such as variable names.

2.2 Bespoke IFC model validation

The typical nature of model validation in the construction industry is well-described in literature (Eastman et al. 2009). There are applications that focus on (a) data completeness/correctness and exchange requirements, sometimes called pre-checking and (b) validation of the model with respect to design requirements. The process is outlined in four phases: rule interpretation, model preparation, rule execution and reporting. For the first part, existing legal documents can be annotated for example using RASE (Hjelseth and Nisbet 2011) to subdivide statements in requirements and applicable types.

As MVDs main aim is to check data exchange requirements in models and the complexity of using mvdXML is high due to required schema knowledge, industry has mostly embraced bespoke and closed solutions. One example is the Solibri Model Checker (SMC)[3]. Part of SMC's popularity arises from its ability to add its own classifications on top of IFC classes (i.e. the model preparation phase), which makes it very forgiving in the quality of input data, and allows people to test expectations on top of that. However, this leniency may be seen as detrimental to high quality OpenBIM adoption as it is ambiguous as to how exactly its tests relate to the IFC schema. In addition, its black-box nature makes it hard to guarantee that tests remain unchanged from version to version, or even persist in the next version, and commercial nature may make it impractical to require from BIM authors. As a parallel, proprietary unit testing frameworks are rare in the software industry. Nevertheless, its popularity demonstrates that there is market demand.

Alternative open source implementations exist, such as the IfcValidator[4] tool, which offers added transparency due to its open-source nature, and stays truer to the IFC schema, but requires precompiled test definitions and a dependency on BIMServer. This requirement increases the barrier to entry for most BIM authors, as BIMServer is not widely used, and to author a test requires the ability to set up a development environment, code compilation, and an understanding of BIMServer and the Java programming language. It is also restricted to data originating within a BIMServer, which can be problematic when data can some from multiple sources and formats from many disciplines. However, its ability to integrate into the model submission pipeline is beneficial for deployment and governance of BIM testing.

It is worth mentioning that there are other types of testing, such as collision and clash testing, auditing paper documentation, as-built verification, and manual design intent checks, but do not overlap significantly with the scope of an MVD, and are not addressed in this paper.

2.3 Software engineering

Within software engineering, the concept of testing and test suites are well established to guarantee that a deliverable is fit for purpose for a particular user. These test suites form part of the quality control procedure, project documentation, issue management process, deployment process, contractual requirements, and are often automated. Unlike the AEC industry, where data testing is fragmented, many software engineering test suites follow the procedures set out by the xUnit[5] framework.

[3] https://www.solibri.com/
[4] https://github.com/opensourceBIM/IfcValidator
[5] https://xunit.net/

One subset of test suite practices is known as Behaviour Driven Development (BDD). The characteristics of BDD are summarized in (Solis and Wang 2011). The first and foremost is that tests are written in a ubiquitous language so that customers and developers speak the same language and can collaboratively edit and review the body of rule definitions.

The software engineering discipline has largely embraced the paradigm of Continuous Integration / Continuous Delivery, meaning that tests are ran automatically and after every change or on a fixed interval.

3. Contribution

In this paper we apply the Behaviour Driven Development practice to a real-world project and construction company (both kept private). In our case, and many others, we used the Gherkin language which provides the benefit of expressing technical BIM concepts in real, human language that document project requirements for multiple stakeholders.

It is well established that mvdXML duplicates the pre-existing standards on model transformations and subset declarations. Further detailing that is not the aim of this paper. What is interesting to conclude from the state of the art is that in the languages described in previous sections there is often a different language for (a) validation and (b) declaration of subsets, but mvdXML aims to do both. Also, in BDD test sets are written in human readable, clearly understandable scenarios so that engineers, managers and auditors all verbally comprehend the rules. The technical focus on data modelling definition and tight coupling with the IFC schema renders mvdXML unsuitable as a basis for discussion with less technically proficient stakeholders. In this paper we elaborate on and discuss our experiments with Behaviour Driven Development (BDD) and testing of building models using the Gherkin syntax.

3.1 Rule translation

In our project, requirements are translated into scenarios written in Gherkin syntax, which are tested against the model. The characteristics of human readability and the ability to build up a complex vocabulary of building requirements render it suitable for application in the construction sector. An example of statements produced by various parties involved in a construction project is outlined in Table 1.

Table 1: Example of rule definitions from an (a) architect (b) lighting designer (c) asset manager and (d) the mvdXML specification.

The project must be geolocated All furniture must be contained in a room All walls must have a material and colour All wall type names must follow the naming convention of “XXXYYY” All wall types must have a fire rating
All spaces must have space lighting requirements All coverings must have a surface style All objects with a material with the name Glass must have an externally defined surface style containing BRDF data
All maintainable assets from the COBie list of assets must have warranty properties All warranty properties must have a point of contact and warranty end date as a minimum All maintainable assets must have an associated document which links to our maintenance document server
All sensors must have at least one port that submits signals

Unlike the typical usage of the MVD standard, these statements are created on a project by project basis, but common ones may be re-used, similar to how design patterns are re-used in

software. These statements may be both internationalised and localised to account for different practices around the world. These statements are organised into scenarios, and parsed by a test runner. The Gherkin language can be parsed by a variety of test runners in a variety of programming languages. Once parsed, the test runner may invoke any software to validate the requirements of the test. It is possible, at this stage, to then generate mvdXML and use an mvdXML validator to check the model. However, it is proposed that writing model queries using a native programming language and an IFC library has benefits over mvdXML. In this paper, we use the IfcOpenShell library and write test implementations in Python, which is interpreted at runtime.

The mvdXML documentation provides a simple example enforcing that sensors have at least one port that submits signals. This can be represented by the single domain specific sentence in Table 1d.

The sentence is unchanged from the everyday domain language of the authors and recipients. The sentence may then be used in test suites and enforced contractually. The implementation in Python using the IfcOpenShell library is 10 lines of code (including line breaks for readability) and is shown below. In contrast, the corresponding implementation shown in the mvdXML documentation is 89 lines of code, excluding line breaks for readability. The Python code also has a certain natural readability, allowing non-programmers to understand what it is describing.

Table 2: The 10 lines of imperative Python code required to check the sample from the mvdXML specification. The equivalent mvdXML requires 89 lines.

```
for sensor in ifc_file.by_type("IfcSensor"):
    if not hasattr(sensor, "IsNestedBy") or not sensor.IsNestedBy:
        assert False
    for relation in sensor.IsNestedBy:
        for port in relation.RelatedObjects:
            if port.Name == "Output" \
                    and port.SystemType == "SIGNAL" \
                    and port.FlowDirection == "SOURCE":
                return
    assert False
```

3.2 Implementation

The BlenderBIM Add-on[6] (Figure 1) was used as an open source, visual BIM authoring and auditing tool. The BlenderBIM Add-on is part of the IfcOpenShell project, and uses IfcOpenShell, an established IFC library to read and write IFC2X3 and IFC4 schemas. Unlike other BIM authoring tools, which historically have emerged from fabrication and CAD industries, the BlenderBIM Add-on runs on Blender, a popular content creation suite in the creative computer graphics industry. This makes it adept at modifying and visualising mesh data, which is exceptionally useful for auditing, as in practice there is almost zero lag in displaying even the heaviest of models. It also has an interactive Python shell to query the model live. The internal data model is also designed specifically to mirror the IFC schema, to minimise data translation during import and export, and treat the IFC data as a native model. Because there is no rigid internal data model such as in the case of the pre-existing BIM authoring tools there is very little reinterpretation needed when opening an IFC file at the expense of the mesh-based nature being unable to mimic the parametric interfaces in such tools such as a

[6] https://blenderbim.org

representation of a wall by two movable end-points and a wall thickness (although parametric functionality in Blender is provided by other add-ons).

This approach to testing project and organisation specific exchange requirements by specifying domain specific Gherkin syntax, alongside a Python implementation in IfcOpenShell was implemented on real world BIM data in Australia to judge its suitability. IFC data was audited from architectural, MEP, landscape, and structural disciplines. The test runner implementation used was Behave[7]. An open-source tool called BIMTester was written as a thin wrapper around Behave. BIMTester would package Behave and the IfcOpenShell library into a single standalone executable, using pyinstaller, to allow users to easily run the tests cross-platform without needing a prior understanding of unit testing or how to install Python or its modules. Behave would return test results in JUnit XML, which is compatible with many available systems that understand xUnit testing. The BIMTester wrapper would then provide additional HTML formatting to the JUnit XML, presenting the test results in a colourful and friendly HTML report to BIM authors. BIMTester is under 150 lines of code.

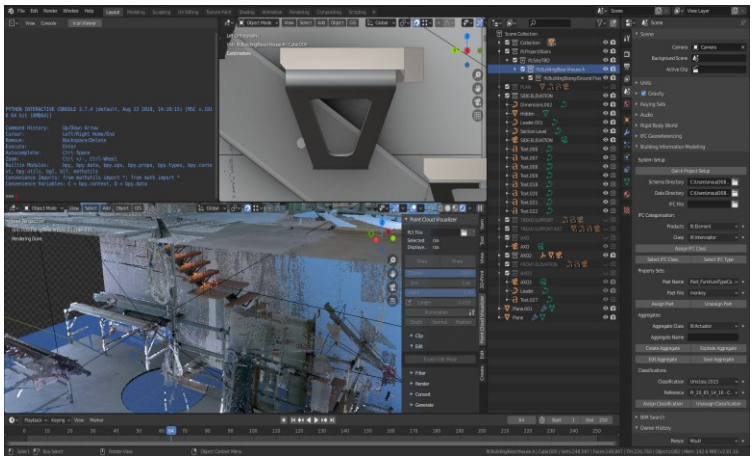

Figure 1: Visualization of a model in the BlenderBIM Add-on with the Blender user interface elements: Python console, renderings, outliner (model decomposition) and properties panel.

The first test suite performed was to check whether an IFC file was of a particular IFC schema. The schema version is often specified in contracts and in digital execution plans, in identical domain language to the sentence used in the test. A domain specific sentence to describe this exchange requirement is given in Table 3a.

Table 3: Two example Gherkin scenarios assessed in BIMTester

Given the IFC file "foo.ifc" Then the file should be an IFC2X3 file
The element {global_id} is an {ifc_class}

The first sentence specifies to the test runner which IFC dataset subsequent tests will be performed on. This is stored in the test environment, which persists across subsequent tests. This notion of persistence allows tests to depend on one another, leading to efficient test execution. Loading an IFC file is a one-liner in IfcOpenShell, as is the check for the schema.

Another fundamental test is to check whether BIM objects are assigned to the correct IFC class. From a contractual and BIM governance perspective, the IFC class determines portions of the digital strategy and execution plans, such as what LOI/LOD applies, which consultant has the

[7] https://behave.readthedocs.io/en/latest/

scope of a particular object, and what objects are to be included in deliverables, such as for the COBie MVD for maintainable assets. From a technical perspective, many of the properties and attributes available to an object in IFC depend on the IFC class, can be used as a guide to check whether the right classification system reference has been applied, and a used as filtered in processes, such as construction sequencing, prioritising clash detection, and scheduling for quantity take-off. Unfortunately, a lot of BIM authoring platforms tightly couple the geometry modelling capabilities with the class assignment, resulting in incorrect classifications. Similarly, a lack of industry knowledge around OpenBIM has led to an over-reliance on default BIM authoring tool IFC class mappings, with little evidence seen of processes in place to ensure correct IFC classes. A domain specific sentence to describe this exchange requirement is given in Table 3b.

In this case, the sentence contains two variables: {global_id} and {ifc_class}. Prior to the models being built, it is not known what IFC classes would be checked, nor what the element GUIDs would be. Therefore, contractually the sentence would be presented as a generic requirement, with an understanding that BIM authors may expect to be audited for any {global_id} or {ifc_class}. An implementation in IfcOpenShell is again one line of code.

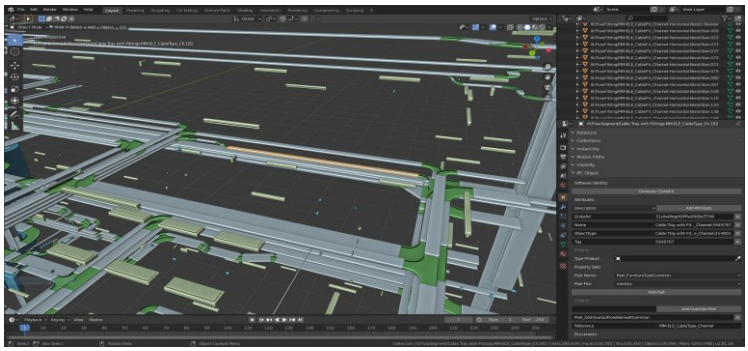

Figure 2: A model from the electrical discipline with objects colour coded by their IFC class value. Small dark blue objects are IfcBuildingElementProxy objects, which are easy to spot.

IFC classes usually require a visual check. Although some can be determined programmatically, such as by testing the aspect ratios of bounding boxes for slabs, columns, and beams, this method is generally less reliable. It is also not sufficient to check for the correct type product, as not all IFC products are typed, and a single generic type may be abused to create multiple different products. Approaches to check correct element class associations based on geometrical properties and proximity to other objects is given in Krijnen and Tamke (2015).

To allow efficient testing, the BlenderBIM Add-on was used with the ability to colour code a model by the IFC class and use drop-downs to select valid IFC classes. IFCs created by BIM authoring programs generally have various patterns in their export, and each firm tends to have their own style based off their internal BIM asset library. Together, these two characteristics allow one to quickly use bulk selection tools based on wildcard names and object attributes to quickly audit hundreds of objects at a time. The Blender modelling package has features for object isolation, hiding, and x-ray vision, allowing for further efficiencies in the test generation process. Objects could be selected in bulk and either have their current IFC class approved, or another IFC class specified as the correct IFC class (Figure 2).

3.3 Continuous Integration

The above results in over a hundred thousand of these tests created in a typical large project prior to construction detailing. As IfcOpenShell builds an indexed list of element global IDs, it is able to very efficiently retrieve any object in the IFC file once it has loaded. Execution of a single one of these tests is in the range of a millisecond, allowing to test the entire project in seconds. The testing bottleneck lies in the disk's capability to load the IFC file through IfcOpenShell, with an IFC file in the hundreds of megabytes taking between 10-20 seconds to load. However, once a file is loaded, the tens of thousands of tests which follow are completed in mere seconds. The entire process taken to spin up a new virtual testing machine with a clean environment, download the latest revision of all of the IFC files, download and install the testing dependencies, and run the tests for half a gigabyte of IFC data and hundreds of thousands of elements, and produce HTML reports for download is approximately 3-4 minutes on a single threaded, standard build server. This full set-up and tear-down process is typical of the process that a software company might have to perform QA. The process is fully automated and executed on a post-receive Git hook.

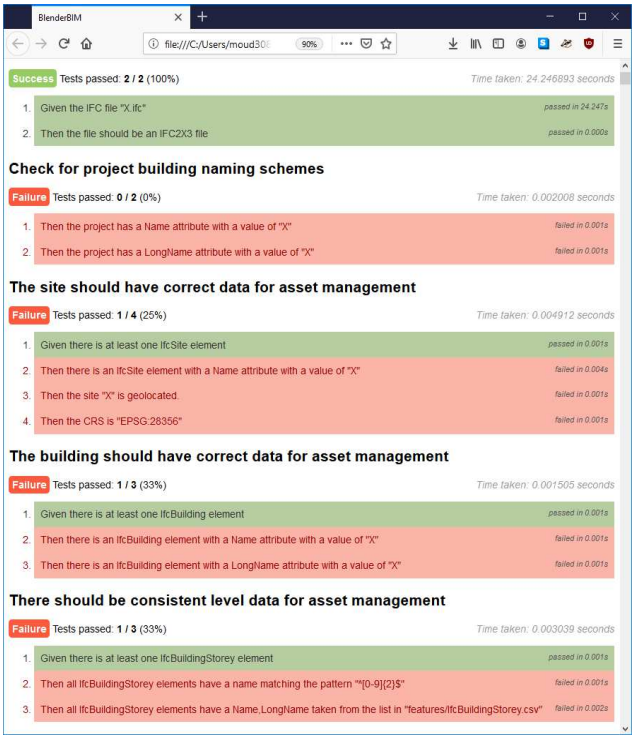

Figure 3: Overview of the test reporting page (somewhat censored). The time taken to load the file (24s) is significantly greater than test execution. The "scenario" syntax of the "given", "when", and "then" prefix is used as a convention, but may be omitted.

Once the test suite is created, it persists throughout all future model submissions. Models are often submitted on a regular basis, so in a week's time a proportion of elements would have been added, removed, or modified. Elements that are modified are already considered by the test suite, and so if the element IFC class changes, the test suite will continue to recheck its class, preventing the possibility of regressions, just as in software engineering. If an element is removed, the BIMTester tool has a purge option, which detects any {global_id} variables in the test suite and deletes all the corresponding tests prior to running the test suite. This is done through simple text file manipulation, as the test suite is a plaintext file, and has a negligible bearing on execution time. The test suite is version controlled with Git, allowing any test

deletions to be undone, for example due to an errant export by a BIM author who may have disabled various categories of elements by mistake. Similar to detecting element removals, element additions are determined by identifying all elements in the model without a corresponding test, which is done using the BlenderBIM Add-on. This has the additional benefit of serving a similar purpose to a model compare, which directs the auditor's attention to new or changed portions of the model. This also means that subsequent audits are much faster, despite this being fundamentally but necessarily a manual process. An initial audit would take approximately 2 days to audit 30 IFC files, but subsequently would only take a few hours.

Once a test is performed and a report produced, it is not always straightforward to correlate the report results (Figure 3) back to a live model to fix the results. For example, not all BIM authoring tools make it easy to select an element by its IFC GlobalId attribute - requiring multiple steps and a specific procedure to be followed. Some OpenBIM viewers do not have this capability at all. Some elements that are composed into an aggregation are hard to select, and some elements are parametrically created (such as arrays) and cannot be individually selected in the native authoring tool. An element may have also been deleted since the audit was performed, resulting in confusion. Guides and plug-ins had to be produced to ensure that this process was smooth.

3.4 Summary and findings

The project documented in this paper had a total of 21 rule sentences implemented. Bridging the nouns in the Gherkin syntax to imperative operations on the IfcOpenShell Python IFC parser took 243 lines of code, averaging 11.5 lines of code per Gherkin syntax. This shows how by embracing existing open source solutions we were able to build an advanced system for automatically and continuously assessing the state of the building model with relation to an evolving body of rule definitions. Executing the rules was almost instantaneous on a test set of 35 files totalling 2.9GB, parsing the data took most time. The imperative implementation in Python and the “batteries-included” nature of its module ecosystem allowed for some interesting side effects, such as being able to cross reference data from an CSV file. This type of integration with external data is currently not feasible in mvdXML. Python also allowed for the usage of math primitives and trigonometry for the validation of latitude longitude. In addition, future work might make use of the fully evaluated geometry provided by IfcOpenShell and the validation n-ary predicates and relationships.

From a political perspective, people were not always used to data being audited to such a detailed level and sometimes the information requires were underspecified to be suitable for automated testing.

4. Conclusion

In this paper we have provided a freely available open source software toolset for the application of rules in the Gherkin syntax to an IFC building model and reporting on the validation outcomes. The coupling of BIM and code opens possibilities within the AEC industry. Several professionals with extensive experience have participated indirectly in the experiments and have since changed their previously negative conceptions of IFC, now considering it as the foundations for seamless, non-deprecating, and lossless data exchange. IFC enabled to take full ownership of data, provide ruthless automation and data integration.

Current implementations of rule checking on BIM models are hindered by the following limitations (a) lack of portability of rules (b) black box implementation (i.e. subject to change, unclear exact behaviour and (c) automatic inference (of quantities, subtypes, relationships)

which is forgiving to end-user on the short term, but will hurt interoperability on the long term due to diverging implementations. We identified that existing ecosystems and frameworks typically have different languages for (a) validation/certfication and (b) model subset declaration. With mvdXML attempting to do both there is not a clear focus, uncertainty in industry and evaluating alternatives is problematic. We have therefore focussed only on model validation by embracing Behaviour-Driven Development approaches.

The trifecta of governance language (to be reused in contractual documents), cheap test creation (the majority of test sentence implementations are under 20 lines of code), and flexible pipeline integration (plaintext definitions and xUnit reporting) suggest that a Gherkin language based approach towards auditing BIM data is useful in the industry. Participants were willing to adapt business processes and easily understood the requirements being audited and imposed. Having automated testing integrated into the model submission process removed all barriers to entry, and the simultaneous provision of open-source test definitions, plain-text test definitions, and cross-platform, small utilities allowed the more curious participants to gain a deeper understanding of OpenBIM, without the need for prior programming experience.

The tests performed only focused on data exchange requirements. However, it is believed that these tests may be extended to cover design intention through the IfcObjective entity or to query spatial requirements. Further research will focus on the interaction of the results of xUnit tests within a BIM authoring tool and connecting to initiatives such as the BCF API and OpenCDE specifications.

References

Bailey, I., Hardwick, M., Laud, A., & Spooner, D. (1996). Overview of the EXPRESS-X Language. In Proceedings of the Sixth EXPRESS Users Group Conference, Toronto, Canada

Eastman, C., Lee, J. M., Jeong, Y. S., & Lee, J. K. (2009). Automatic rule-based checking of building designs. Automation in construction, 18(8), pp. 1011–1033.

Hjelseth, E., & Nisbet, N. (2011). Capturing normative constraints by use of the semantic mark-up RASE methodology. In Proceedings of CIB W78-W102 Conference.

Krijnen, T., & Tamke, M. (2015). Assessing implicit knowledge in BIM models with machine learning. In Modelling Behaviour (pp. 397–406). Springer, Cham.

Liu, H., Gao, G., Zhang, H., Liu, Y. S., Song, Y., & Gu, M. (2019). MVDLite: A Light-weight Representation of Model View Definition with Fast Validation for BIM Applications. arXiv preprint arXiv:1909.06997.

Pauwels, P., Krijnen, T., Terkaj, W., & Beetz, J. (2017). Enhancing the ifcOWL ontology with an alternative representation for geometric data. Automation in Construction, 80, pp.77–94.

Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., & Reijers, H. A. (2011). Imperative versus declarative process modeling languages: An empirical investigation. In International Conference on Business Process Management. pp. 383–394. Springer, Berlin, Heidelberg.

Solis, C. and Wang, X. (2011). A Study of the Characteristics of Behaviour Driven Development. In: 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications, Oulu, 2011, pp. 383–387.

Zhang, Chi, Jakob Beetz, and Matthias Weise. "Interoperable validation for IFC building models using open standards." Journal of Information Technology in Construction (ITcon) 20.2 (2015), pp.24–39.