Alternate Means of Digital Design Communication

An Investigation of Design Data Representation, Persistence and Transaction for the Architecture, Engineering and Construction Domains

Dimitrie A. Stefanescu

Submitted in partial fulfilment of the requirements for the degree of Architectural Design MPhil/PhD

The Bartlett School of Architecture,
University College London
Supervisors: Dr. Sean Hanna, Prof. Bob Sheil, Dr. Robert Aish
I, Dimitrie Stefanescu, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

____________________________________
Signature
Abstract

This thesis reconceptualises communication in digital design as an integrated social and technical process. The friction in the communicative processes pertaining to digital design can be traced to the fact that current research and practice emphasise technical concerns at the expense of social aspects of design communication. With the advent of BIM (Building Information Modelling), a code model of communication (machine-to-machine) is inadequately applied to design communication. This imbalance is addressed in this thesis by using inferential models of communication to capture and frame the psychological and social aspects behind the communicative contracts between people.

Three critical aspects of the communicative act have been analysed, namely (1) data representation, (2) data classification and (3) data transaction, with the help of a new digital design communication platform, Speckle, which was developed during this research project for this purpose. By virtue of an applied living laboratory context, Speckle facilitated both qualitative and quantitative comparisons against existing methodologies with data from real-world settings.

Regarding data representation (1), this research finds that the communicative performance of a low-level composable object model is better than that of a complete and universal one as it enables a more dynamic process of ontological revision. This implies that current practice and research operates at an inappropriate level of abstraction. On data classification (2), this thesis shows that a curatorial object-based data sharing methodology, as opposed to the current file-based approaches, leads to increased relevancy and a reduction in noise (information without intent, or meaning). Finally, on data transaction (3), the analysis shows that an object-based data sharing methodology is technically better suited to enable communicative contracts between stakeholders. It allows for faster and more meaningful change-dependent transactions, as well as allow for the emergence of traceable communicative networks outside of the predefined exchanges of current practices.
Impact Statement

The work presented in this thesis forms part of a broader context pertaining the digitalisation of work in the AEC (Architecture, Engineering and Construction) domains. This has been recognised by the European Union’s Horizon 2020 Research and Innovation Program, which funded this research as part of the Innochain Network (Marie Skłodowska-Curie Actions).

The research presented in this project has already spun-off a follow-up grant funded by InnovateUK (part of UKRI), “AEC Delta Mobility”, whose goal is to research and define an industry standard for design change specification that allows faster, and more agile, “delta” updates to replace file-based exchange mechanisms that are currently prevalent in the AEC industry. Furthermore, the research instrumentation developed as part of this project has been used as a technological base for research in participatory urbanism at the Future Cities Laboratory at the Singapore ETH Centre. Succinctly put, from an academic perspective, the findings of this project can be used to inform further research in digital design collaboration, construction and project management, as well as revitalize scholarship in the realm of data interoperability in AEC.

Outside academia, Speckle, the research instrumentation developed, continues, at the time of writing, to exert a growing influence on the professional practice of architects, engineers, and other stakeholders involved in the design and construction process. It is incorporated by a number of international AEC companies at the core of their digital transformation efforts. Amongst the most prominent are HOK, SOM, Arup, Sasaki, BVN, Aurecon, Dialog Design, Grimshaw, and many others.
Acknowledgments

This research has been conducted at the Bartlett School of Architecture, UCL, within the InnoChain Early Training Network supported by the European Union’s Horizon 2020 Marie Skłodowska-Curie grant agreement No. 642877.

This project would have not been possible without my supervisors, Sean Hanna, Bob Sheil and Robert Aish, whose guidance was invaluable. Similarly, I owe an huge debt of gratitude to Martin Tamke and Mette Ramsgaard-Thomsen who enabled the InnoChain project in the first place, as well as to the generosity of the persons that made this research network possible. Most importantly, to Luis Fraguada, who was a productive and calming presence at McNeel Europe, and to Giovanni Betti, the discussions with which could never end, at HENN Berlin.

Further thanks go to my mother (who showed me the way to Garfinkel, Grice, Sperber and Wilson), my father, my partner and my closest friends. Without your patience and support, this project would have not been done.

London, October 2019
Table of Contents

Abstract ................................................................................................................................. 2
Impact Statement .................................................................................................................... 3
Acknowledgments .................................................................................................................. 4
Table of Contents .................................................................................................................. 5
List of Figures ........................................................................................................................ 9
Glossary ................................................................................................................................. 14

1. Introduction ....................................................................................................................... 16
   Data representation ............................................................................................................... 17
   Data Classification ................................................................................................................ 19
   Data Transaction .................................................................................................................. 20
   Methodology and Research Instrumentation ...................................................................... 21

2. Literature Review ............................................................................................................. 23
   2.1 Introduction .................................................................................................................... 24
   2.2 Differentiation and Divergence ..................................................................................... 25
      2.2.1 Industrial and Scientific Specialisation ................................................................. 25
      2.2.2 Professions and Conflict ...................................................................................... 27
      2.2.3 Modernity, Transparency and Trust ...................................................................... 29
   2.3 Design & Convergence ................................................................................................. 33
      2.3.1 Metaphors and Conceptual Displacement ............................................................ 33
      2.3.2 Wicked Problems ................................................................................................. 36
      2.3.3 Solving Wicked Problems .................................................................................... 37
   2.4 Design and Communication .......................................................................................... 42
      2.4.1 Communication Models ....................................................................................... 47
      2.4.2 Code model of communication ............................................................................ 48
      2.4.3 Inferential models of communication .................................................................... 49
   2.5 State of The Art: Digital Communication & Design ..................................................... 54

3. Methodology ...................................................................................................................... 63
   3.1 Living Laboratory .......................................................................................................... 64
   3.2 Tackling a Wicked Problem ......................................................................................... 66
5.6.2 Staging Workflows ................................................................. 127

5.7 Observed Classification Patterns & Performance ................................ 129
  5.7.1 Model Subdivision & Reassembly ........................................... 129
  5.7.2 Correlating Sources and Receivers ........................................ 131
  5.7.3 Storage Performance ............................................................ 132

5.8 Conclusion .................................................................................. 135

6. Data Transaction ............................................................................ 137
  6.1 Nextness & Sequentiality ............................................................ 138
  6.2 Optimising Transactions: Differential Updates ............................... 143
    6.2.1 Differential Updates ............................................................. 143
    6.2.2 Observed Performance ......................................................... 145
    6.2.3 Theoretical Best & Worst Cases ........................................... 149
  6.3 Assembling the Network (Instrumentation) .................................... 150
    6.3.1 Users .................................................................................. 151
    6.3.2 Clients ................................................................................. 152
  6.4 Information Propagation .............................................................. 154
  6.5 Case Studies ............................................................................... 157
    6.5.1 Solution Space Precomputation and Mass Customisation .......... 157
    6.5.2 Design and Visualisation Workflow ....................................... 162
    6.5.3 Design and Façade Engineering ............................................ 163
    6.5.4 Summary ............................................................................. 166
  6.6 Conclusion ............................................................................... 167

7. Discussion ...................................................................................... 169
  7.1 Schemas and Standards ............................................................... 170
  7.2 Classification and Curation .......................................................... 173
  7.3 Digital Transactions & Communication ........................................ 177
  7.4 Data Ownership .......................................................................... 180

8. Conclusion ..................................................................................... 183

9. Bibliography .................................................................................. 187

Appendix A: Tooling .......................................................................... 198
  Tooling History .............................................................................. 199
Speckle A........................................................................................................................................199
Speckle B........................................................................................................................................200
Speckle C........................................................................................................................................202
Speckle D – Speckle.Works, the open source data platform for the AEC industry ..................204

Technical Overview & Code Documentation .................................................................................205

Appendix B: Industry Exchanges .................................................................................................206

Data Communication in AEC / 1st Speckle Meeting.....................................................................207
Data Communication in AEC / 2nd Speckle Meeting...................................................................217
Data Communication in AEC / 3rd Speckle Meeting ................................................................220
Feedback from C.F. Møller Architects.........................................................................................222
Speckle Survey...............................................................................................................................226

Appendix C: Impact .......................................................................................................................229

Ongoing Research.........................................................................................................................229
Industry Adoption.........................................................................................................................230
Online Engagement Metrics.........................................................................................................230
List of Figures

Figure 1: Asymmetric reflections of meaning (left); The emergence of a new domain (right) ............................................................34
Figure 2: Ecological phylogeny displaying ever increasing branching (left) vs. Cultural phylogeny showing both branching and merging (right) ................35
Figure 3: "Design alternates between figure and representation" (Hanna, 2014). ....37
Figure 4: Shannon’s diagram of the communication model (Shannon, 1948). ........48
Figure 5: Activity network diagram (left); Message exchange scenario diagram (right) (de Vries, 1995, p. 472) ..........................................................................................57
Figure 6: Workflow diagram for Project Bank describing the process of branching and merging in a two-user scenario (Aish, 2000) .................................................................58
Figure 7: Schematic overview of the Speckle platform: a central REST API, implemented by a server, and consumed by a host of other client applications. ........................................................................................................67
Figure 8: Above: the specification entry for the Mesh geometric primitive. Below: the implementation in C# of the Mesh class, automatically generated from the specification above. ..........................................................................................77
Figure 9: The construction of multi-schema objects in the parametric modelling software, Grasshopper. Reproduced with the permission of the author (Poinet et al., 2018) ........................................................................................................80
Figure 10: Ontological revision process: objects can be packed, unpacked and re-defined in various contexts by their users. .................................................................81
Figure 11: Example diagrammatic representation of user defined objects. ........81
Figure 12: (Code block) Pseudo-code implementation of the encoding method for a SpeckleAbstract object ...............................................................................................84
Figure 13: A graph showing the intermediate representation of a BHoM Bar object, as it was cast into a SpeckleAbstract DTO .................................................................86
Figure 14: A diagrammatic representation of the reference tree of a given BHoM Bar object. The black circles represent object properties, the large grey circles represent embedded objects, and the arrows indicate nested references. ....87
Figure 15: Diagram showing a generalised case of recursive references with a given object. Left: deeply-nested references; Right: simplified example, showing the child nodes directly referencing their parent. .................................................................88

Figure 16: Evolution of the ggIFCGrid object representation: the right hand side version is the original, whereas the left hand side version is the optimised version. ................................................................................................................................................89

Figure 17: Comparative table showing the depth of inheritance and class coupling metrics for three existing object models. ..............................................................................................................................................90

Figure 18: Comparative analysis between the average tree depth and size of BHoM- and ggIFC-derived SpeckleAbstract objects. .................................................................................................................................................91

Figure 19: Comparative analysis table of object depths and sizes. ........................................................91

Figure 20: Table representing the number of object types that have been customised by end users. .........................................................................................................................................................94

Figure 21: Table view of an imaginary object model kit, which shows the demarcation between the class definitions and their actual conversion implementations in separate CAD software applications. ................................................................................................................................96

Figure 22: (Code Block) Signature of the ToNative and ToSpeckle functions........97

Figure 23: Left, the tangled communication patterns of the design process; right, the centralised “one model” approach.........................................................................................................................104

Figure 24: The contemporary equivalence between a technical model of communication and a social one. .........................................................................................................................................................104

Figure 25: Diagram showing the typical ways design information is cast into separate models for specific analysis tasks, and the subsequent feedback of the results in the design model.................................................................................................................................107

Figure 26: Speckle platform high-level architecture..................................................................................111

Figure 27: Decentralised network of Speckle instances and clients. .......................................................111

Figure 28: Implementation of the CreateObject POST endpoint. ..............................................................112

Figure 29: OpenAPI specification of the CreateObject POST endpoint.................................................113

Figure 30: The Speckle database model, showing a simplified view of the collections (tables) and relationship between them...........................................................................................................114

Figure 31: Grasshopper Speckle Sender (left) and Speckle Receiver (right).................................116
Figure 32: Rhino Sender (dark blue) and Receiver (teal).

Figure 33: Diagram showing a feedback loop between two users working on the same geometry: A, via stream 1, sends to B an object. B modifies the object, and sends to A, via stream 2, her or his proposed changes.

Figure 34: Expanded diagram showing the interactions between multiple actors and SpeckleStreams, and the emergence of a communication network in the shape of a directed graph.

Figure 35: Shows the process behind Dialog’s building programming optimisation workflow. Image rights: Mark Cichy & Dialog (2018).

Figure 36: Still from Simon Yorke's video describing the workflow enabled by Speckle. Image rights: Simon Yorke & Aurecon (2018).

Figure 37: TU Delft SimAud 2018 guided exercise workflow map (above) and actual resulting model (below).

Figure 38: Diagrammatic representation of the evolution of the communication flows during the unguided exercise of the workshop.

Figure 39: Distribution of senders (black), receivers (blue), and both (red) per model. Vertical axis: number of models (logarithmic scale).

Figure 40: Distribution of sources per model.

Figure 41: Breakdown of sources & receivers by application type.

Figure 42: Total storage space performance in a multiple sub-classification scenario on an example model.

Figure 43: Total storage space performance in a multiple sub-classification scenario on a model coming from a real life project.

Figure 44: Sample storyboard showing conception stages in the engineering process, from early stage design (sketches, hand calculations and simple analysis models) through the middle stages, where the BIM model is actually composed, and finally to final, detailed design stages.

Figure 45: Left: Architect-Engineer workflow during the design and analysis of a complex interior staircase project. Right: Workflow for a bid costing process undertaken by a developer and its subsidiaries. In red, is the measurement based process (quantities), and in blue the activity based process (site planning & construction management).
Figure 46: Comparison table between transactions in a differential update approach (Delta) and a bulk sharing approach. .................................................................146

Figure 47: Transaction breakdown: green represents newly added objects, red removed, and grey unchanged. .................................................................147

Figure 48: Historical analysis of a stream’s transaction record........................................148

Figure 49: Transaction history for Stream E. Green represents newly added objects, red removed, and grey unchanged. Figure 50: (Below) Expanded table showing transactions 32 and 36. .................................................................149

Figure 51: Message diagram showing the hypothetical flow of information between three clients (A, B, and C), as mediated by the Speckle Server. .........................156

Figure 52: Screenshot of the online interface developed for citizen engagement, based on a pre-computed parametric model. From Knecht et al., 2019...............159

Figure 53: Simple Wood Goods example application: web interface (left), customised real product (right). Image rights: Matthew Swaidan, 2018. .................................160

Figure 54: Nstance web application (left), and corresponding ordered product (right). Image rights: nstance.ro, 2018.................................................................160

Figure 55: Process diagram summarizing the information flow between end-user, model, and manufacture.................................................................161

Figure 56: Diagram representing the communication network between the stakeholders. Image rights: Fernando Ruiz/Arup, 2018. ..................................................165

Figure 57: Data transaction case studies summary table .................................................166

Figure 58: Above: Similar technical code models of communication enabling two different processes in the context of digital design communication. Below: relationship between the communication channels and their supportive (centralised) technical embodiment.................................................................176

Figure 59: Activity network graph generated for a set of data streams from a live project. Left: full graph; middle: grouped by users; right: grouped by documents. Image credits: Paul Poinet, research done at UCL BCPM under the AEC Deltas InnovateUK grant. .................................................................179

Figure 60: Different activity network, with icons symbolising different software. Image credits: Paul Poinet, research done at UCL BCPM under the AEC Deltas InnovateUK grant.................................................................179

Figure 61: A typical fabrication process (Sacks et al., 2010).
defined.

Figure 62: Multi-server communicative network scenario.
Glossary

**AEC**: Architecture, Engineering and Construction.

**AECO**: Architecture, Engineering, Construction and Operation.

**API**: Application Programming Interface.

**BIM**: Building Information Modelling.

**Brep**: Boundary Representational Object.

**C#**: Widely used programming language developed by Microsoft.

**CRUD**: Create, Read, Update, Delete.

**DAC**: Discretionary Access Control.

**DTO**: Data Transfer Object.

**FEM**: Finite Element Method.

**GUID**: Globally Unique Identifier.

**Git**: Collaborative version control system for (code) files.

**HCI**: Human Computer Interaction.

**Hestia**: The name of the public Speckle server.

**HTML**: Hyper Text Markup Language.

**IFC**: Industry Foundation Classes.

**JSON**: JavaScript Object Notation.

**JWT**: Json Web Token.

**MEP**: Mechanical, Electrical and Plumbing [Model].

**.NET**: Programming framework developed by Microsoft.

**NoSQL**: Not Only SQL.

**OpenAPI**: Industry standard open specification standard for REST APIs and

**POCO**: Plain Old C# Object.

**Polycurve**: A curve consisting of two or more sub-curves of different types.

**REST**: REpresentational State Transfer.

**SQL**: Structured Query Language.
**Speckle**: Software framework developed during the course of this project as research instrumentation.

**UUID**: Universally Unique Identifier (see GUID).

**VDC**: Virtual Design and Construction.

**VDI**: Virtual Desktop Infrastructure.

**WS**: Web Sockets
1. Introduction

Design, in general, has to solve ill-defined, or "wicked", problems, the understanding of which is concomitant to the act of their resolution. In the words of Rittel and Webber, “the information needed to understand the problem depends on one’s idea of solving it” (Crowley and Head, 2017; Rith and Dubberly, 2007; Rittel and Webber, 1973). As such, the act of design does not have a definitive stopping rule, and neither can its output be judged by a binary evaluation of good vs. bad. Most importantly, the stakeholders involved in the design process do not necessarily have a set of shared values. Essentially, design can be seen as an iterative act that aims to reduce uncertainty at an ontological level (Hanna, 2014) by simultaneously searching for the appropriate problem representation and resolution.

The main goal of this project, as set out in the original InnoChain call (Work Package 3: Design Communication, ESR05: Alternate Means To Communicate Measure), is to analyse how complex digitally based design can be communicated and collated internally, within a design team, and externally, with the various stakeholders involved in the design process. Literature establishes that communication and dialogue are the base for solving wicked problems, as through these means shared understanding can be constructed amongst the actors involved in the (design) process (Dawes et al., 2009; Lawson, 2005; Roberts, 2000; Conklin, 2005; Walz et al., 1993; Bechky, 2003). In other words, shared understanding can be construed as a set of matching ontological representations of meaning that gradually emerge through a process of conceptual displacement (Koestler, 2014; Schön, 2011, 1991). Communication and dialogue are increasingly reliant on digital means, nevertheless they cannot be fully analysed through the lens of a purely technical model. As such, in the context of this thesis, communication is understood as a transactional phenomenon that has both a technical or mechanistic manifestation (digital) and, as well, an intrinsic psychosocial component. The former approach corresponds to the Shannon-Weaver model of communication (Shannon and Weaver, 1963, 1948), whereas the latter draws from inferential models developed in the philosophy of natural language communication (Grice, 1991; Sperber, 1995; Wilson and Sperber, 2008).

Currently, the design process exhibits a certain friction in its communicative processes that is perceived by the stakeholders in the Architecture, Engineering and Construction (AEC) industry as holding back the application of the full potential of new digital technologies, contrary to the developments in other domains of the global economy (Barbosa et al., 2017). In part, this research is based on the hypothesis that this friction results from the technical limitations of the software tools, standards and
methods in use. Nevertheless, one cannot dismiss the human aspect involved in the communication process. Particularly, this unfolds into issues around authorship translated into disciplinary and professional boundaries (Giddens, 1991; Abbott, 1992), and materialises into issues of trust stemming from disembodied data (Tsoukas, 1995). Disembodied data, as discussed at length in Section 2.2.3, is the result of separating information from meaning, or intention. To this extent, the investigation will not be limited to purely software aspects of the problem, but out of the necessity to offer a complete picture, will include the psycho-social context in which digital communication unfolds, is shaped by, and simultaneously shapes.

The design process, as currently unfolding in the contemporary societal milieu—coupled with the design problems’ complexity and scale—requires a large array of stakeholders, from both technical backgrounds as well as from outside specialist disciplines, to engage in a continuous (but to a certain extent temporally bounded) act of communication and creation in order to complete a given design assignment. These networks are seen as the basis for resolving wicked problems (Weber and Khademian, 2008). Nevertheless, at the beginning of a design project, one cannot assume the existence of a defined ontological base upon which stakeholders may interact. Furthermore, one cannot fully know in advance who the stakeholders will be and in what parts and capacities they will be involved in the resolution of the problem—the communicative topology of the network, as well as its informational needs, is not fully known. As such, communication in design cannot rely fully on known data representation and classification patterns, and neither on predefined interaction networks.

Consequently, this research project aims to investigate critical aspects of design communication, whereby communication is understood as having both technical and social dimensions that reinforce each other (Garfinkel and Rawls, 2006). Following that, at the beginning of the design process, one is bounded by an incomplete definition of the given problem and by the fact that the relationships between actors need to first emerge before subsequent shared ontologies are defined. Three main research directions have been selected: (1) data representation and (2) data classification juxtapose different approaches to ontological models of design objects and, finally, (3) data transaction looks into the mechanisms of interactional exchange and how they change the nature of design communication.

**Data representation**

Currently, design data exchange is dominated by the Industry Foundation Classes (IFC) object model, which positions itself as the official interoperability standard of the AEC industry, and has been in active development since 1994 (Froese, 2003;
Plume and Mitchell, 2007; van Berlo et al., 2012). It is widely used throughout all stages of the design and build process and underpins most, if not all, Building Information Modelling (BIM) techniques, regulations and practices. Similar to Cyc, an artificial intelligence project, started in 1984 and still ongoing, that attempts to assemble a complete ontology spanning the basics of “how the world works” (Lenat et al., 1985; Lenat and Guha, 1991), IFC classes aim to specify a complete vocabulary and grammar of all elements (both high level—walls, doors, etc.—as well as low level—points, lines, etc.) that are used in architectural design, engineering and construction (Hamil, 1994).

Nevertheless, many large engineering and design companies have been developing in-house alternative design data exchange standards positioned as internal replacements for the official IFC standard, rather than as additions or extensions, to better suit their own internal needs. These developments result from the problematic nature of IFC, whose scope, while vast, is limited, and extensions of it are not easily shared: similarly to Cyc, it is bound to be (forever) incomplete and difficult to evolve (Bertino et al., 2001). These ad-hoc informed standards have scopes that vary from a single-project base to a company-wide norm. To a certain extent, the emergence of such standards results from the need to further articulate domain- and organisation-specific knowledge that accumulates over time: most observed examples underpin computational techniques, workflows and methodologies that quintessentially represent a given organisation market advantage, or a given domain’s internal language constructs that allow for its effective functioning (Giddens, 1991; Kuhn, 1996).

Regarding the representation of information, modern computer science techniques allow for transparent and reflective approaches to constructing higher level standards. Based on self-describing structures (i.e., XML or JSON), only a low level, primitive standard is enforced thus allowing for the bottom-up emergence of expressive, ad-hoc defined object models whose definition, interpretation and validation are relegated to the communicants, rather than the transmission medium itself. The simultaneous readability of data by both machine and human allows for the development of rigorous informational exchanges and applications that consume said data.

The emergence of multiple deregulated project- or organisation-based standards hints at a larger question than IFC’s erosion of the one standard approach. To a certain extent, ontological uncertainty has been embraced and enabled by computer science through the development and increasing adoption of self-describing structures for object representation. Within AEC, it is not yet known what is the best
model of data representation that can serve communication and collaboration needs.

Q1: Consequently, one of the central research aims of this thesis is to identify the limitations (and advantages) of a schema agnostic, self-describing and composable object model compared to an ontologically complete “one standard” approach.

**Data Classification**

Following the questioning of data representation, this research project will look at existing methods of data collation and classification pertaining to existing digital collaboration practices in design. Currently, AEC software is reliant on monolithic data structures that may be traced to the implementation choices of the data representation standards. BIM practices currently advocate a collaborative approach centred around the idea of “one model”, or one single source of truth, that contains, synchronises and validates the information produced by all technical stakeholders involved in the design process. The “one model” approach is encouraged by the mandatory inter-relation of building objects which is prescribed by the data representation standard in use or by the architecture of the software used. For example, as of the latest IFC release, “round-tripping”, or sending and receiving of model information, is stated to be out of scope and, in general, not supported (buildingSmart, 2016a, 2016b, 2016c; Bentley Systems, 2008; Thein, 2015).

Nevertheless, when confronted with the reality of communication in a multidisciplinary design process, the monolithic model prevents natural dialogue to emerge due to its high processing cost when used as a communication vehicle. This problem is mostly visible at the early design stage but is not limited to any particular point in time of the design process. The symptoms of this problem, revealed through industry exchanges undertaken within the research project, manifest as the proliferation of “residual models” (Scheurer and Stehling, 2011), i.e. files containing partial models with selective information that responds to various stakeholders’ specific needs at one point in time. The emergence of lean data exchange workflows can be attributed to the fact that communication tends towards an optimally relevant state (Sperber, 1995; Wilson and Sperber, 2008) in a specific given context. Relevance theory, based on Grice’s inferential model of communication, suggests that communication effort is minimised through the reliance on interpretation and “expansion of meaning” at the receiver end (Grice, 1991; Lindblom, 2001), and through contextually-based informational enrichment mechanisms (Sperber, 1995; Wilson and Sperber, 2008).

Q2: Consequently, existing standards of data classification in digital design
need to be questioned. Specifically, the investigation will focus on what extent design data can be compartmentalised but still maintain logical coherence through an analysis of the advantages and limitations of an object-centred classification approach to digital design data as opposed to a file-centric collaboration methodology.

**Data Transaction**

Data is not the only ingredient necessary for a communication contract to be enabled at a social level, and data alone does not suffice to allow for the emergence of shared understanding (Aish, 2014; Aish and Fleming, 1977; Bechky, 2003; Strathern, 2000; Tsoukas, 1997). As such, the following investigation focuses on another critical aspect of digital design communication, namely its transactional nature: specifically, how data is exchanged. In this regard, literature establishes *nextness* and *sequentiality* as the key variables describing communication contracts (Lindblom, 2001; Sacks et al., 1974). Nextness, or adjacency, ties participants into a cursive dialogue by requiring that informational transactions happen within a certain time (and potentially spatial) limit. The latter, sequentiality, looks at the order in which communicative transactions occur and is especially important in an asynchronous digital environment.

Existing BIM collaboration workflows, due to the nature of existing standards for data classification and representation as well as current software limitations, do not lend themselves towards spontaneous transactions with *actual* design data. Thus, within the digital design process, spontaneous dialogue is delegated to classical means of communication such as chat applications (i.e., Slack), email and attachments composed of residual models, screenshots, etc. and can be seen as introducing friction by not being intimately tied with design data.

Conversely, literature reveals a large number of custom workflows being developed to serve the interoperability needs of large-scale design projects (Bhoosan, 2017; Deutsch, 2017; Goldup et al., 2017; Heumann and Mullenix, 2014). Many of these workflows show a common effort to achieve responsiveness, or nextness. They achieve this mostly through unpacking design information into its most relevant parts for the task at hand (i.e., centrelines and section for a beam element), thus minimising the amount of information that is transacted.

Q3: Complimentary to data representation and classification, data transaction requirements for digital design communication are not yet fully understood or supported. *Consequently, this research project looks at what are the
advantages and limitations of an object-based collaboration methodology as opposed to an file-based one in regards with enabling the key requirements of a communicative contract, namely nextness and sequentiaity.

**Methodology and Research Instrumentation**

In order to be able to answer the questions outlined above, this research project necessitated software design development so as to be able to counterpose existing approaches to a feasible alternative. This resulted in the software platform named Speckle, which served as base research instrument throughout this project.

Firstly, the Speckle platform can be described as being schema agnostic: it does not have a standardised ontology per se, but rather a small set of user-defined, composable object models that can be swapped in and out, and used in together. Secondly, Speckle is object-centric, as opposed to file-centric: instead of persisting data in monolithic blocks, it stores each object individually (and immutably) and allows for overlapping groupings thereof. Third, Speckle is embodying data: as opposed to existing approaches, where files are just “shared” and there is no overview of who is consuming the information, and to what effect, Speckle traces the communicative network and aims to inform end-users of their transactions and their implications.

Speckle enabled a theoretical, technical and applied analysis into the three research questions outlined above. The collaborative network within which this research project was undertaken, InnoChain, consisted of both academic and industrial partners. This network served as the seed for a living laboratory which grew throughout the project to include many other industry participants. As such, this setting served both as the basis for the technical development and testing of Speckle, as well as the pool from which the quantitative and qualitative data required to answer the questions outlined above was gathered from.

Followingly, in the *Literature Review* chapter, the key preliminary issues raised by the research questions will be tackled. Broadly put, the analysis will examine the act of design as a communicative act. Specifically, it investigates the causes behind the divergence of disciplines and how inter-professional communication unfolds, as well as how disciplines re-converge as dynamic networks of stakeholders when tackling wicked (design) problems. Further, an analysis of existing communication models – both technical and social – frames, when coupled with the state of the art in digital design communication, the theoretical context in which the counterpoint to existing approaches to digital collaboration in AEC was formed and assessed.
2. Literature Review
2.1 Introduction

Design is a broad term. It is simultaneously a detailed process that involves a wide array of disciplines and, compounding the problem, affects an even larger number of actors that do not necessarily fit within the categorisation of a specialist profession. The given research topic, digital design communication, adds another dimension to the scope of the investigation, namely technology and its shaping (while being shaped by) the means towards which it is applied. All of these parts are well established fields of research in their own and this research thesis will not exhaustively delve into them; the aim within this chapter is to cross-reference key concepts, ideas and practices from the fields of sociology, communication, design, and computer science to describe a relevant context for this investigation. The value of communication lies in the connections it enables between participants; similarly, the value of this chapter lies in the transversal linkages it makes between communication, digital technologies, and the design industries.

The following review is structured in three main parts, namely Divergence and Specialisation (Section 2.2), Design and Convergence (Section 2.3), and Design and Communication (Section 2.4). Section 2.2 aims to explain the process of modern societal differentiation and introduce the resulting ontological divergence of disciplines (and corresponding stakeholders) and its implications on communication. Subsequently, Section 2.3 reviews scholarly work that, in a design context, investigates the complexity of multi-stakeholder interaction and the way shared understanding emerges through iterative ontological displacement. This is followed by an analysis of existing communication models in Section 2.4. Finally, Section 2.5 looks at existing work in digital design communication, starting from early work done in the 1970s to current standards emerging from practice as well as industry groups and links this research to previous applications in collaborative design software and current gaps in knowledge.

Communication is an essential activity that permeates the design process in all its aspects, from ideation to materialisation. Contrary to popular representations, design is not just an isolated act of creation—or a manifestation of genius on behalf of one single author—but an inherently collaborative process in which a great number of people, with vested interests and from various disciplines, participate. These people—the design stakeholders—through their interaction both define as well as resolve design problems.

For example, a small single-family home project will involve as design stakeholders a client, an architect, a construction company and the local planning authority. Nevertheless, even this simple scenario—involving at first sight just four
stakeholders—has hidden interactions that, normally, are not associated with the design process, yet have a huge influence on it: the client’s interactions with his bank, which finances the whole endeavour and decides the overall project time, the planning authority’s regulations regarding specific usage of local materials and their impact on the budget and on the choice of construction company, etc. For larger projects, such as an apartment block, office building or train station, the number of involved stakeholders and their interactions grow as more specialised disciplines are needed to provide expertise: structural and mechanical engineers, environmental analysts, specialised modelling consultancies, infrastructure planners, financial consultancy companies that orchestrate budgets, contractors and manufacturing companies, etc. Not only does the technical group of stakeholders grow, but so does the non-technical one: communities, business and political actors have the right to be informed and, to a certain extent, have a say, in the development of a project that affects their surroundings and livelihoods.

All the actors mentioned above exchange information and communicate. Their actions shape the built environment, piece by piece—project by project. This degree of high technical specialisation and social complexity allows for ever greater and more complicated problems to be tackled, yet it does not come without friction, conflict and errors which ultimately may lead to unsatisfactory design problem resolutions. In order to understand these issues and how they reflect and influence communication in the design process, it is necessary to look into the mechanisms behind the functional specialisation and differentiation of the current modern society: where do these stakeholders come from, and how do they interact?

2.2 Differentiation and Divergence

2.2.1 Industrial and Scientific Specialisation

Functional differentiation is a process of progressive diversification of societal systems that explains the transition from primitive hunter-gatherer societies to agricultural ones, and subsequently modern society. The division of labour, as an analogue process, is a well-established concept with a long history of intellectual debate as to its positive and negative aspects. At its core lies the simple notion that individuals will perform better at a given task—with greater efficiency and producing higher quality output—if able to specialise in a specific aspect of the problem (Durkheim, 1997; Kant, 2002; Smith, 2017). In the words of Kant,

“All trades, handicrafts, and arts have gained through the division of labour, since, namely, one person does not do everything, but rather each limits himself to a certain labour which distinguishes itself markedly from others by
its manner of treatment, in order to be able to perform it in the greatest perfection and with more facility. Where work is not differentiated, where everyone is a jack of all trades, the crafts remain at an utterly primitive level.” (Kant, 2002).

Within the contemporary domain of design and engineering, this industrially driven differentiation is easily apparent in the construction stage of project: usually led by a general contractor, the building process is split and delegated to a network of sub-contractors, sub-sub-contractors, and manufacturers whose responsibility is bounded to their area of expertise (e.g., foundation excavation, rebar placement, window mounting, electrical systems, etc.). Let’s take the example of a suspension bridge. The mathematical and physics knowledge (calculus, load distribution, cable tensions, vibration analysis, etc.), coupled with the manufacturing expertise needed to produce the necessary building components with the correct physical specification (steel cables, towers, anchorage, etc.) and the operational science of constructing such an engineering project, are beyond the powers of one single individual: each aspect involves its own domain and requires years of study and practice to master.

Industrial specialisation and labour division explain only one part of the diversity of stakeholders in the design process. The suspension bridge example hints at a different dimension to the process of differentiation, namely one that operates at the level of scientific knowledge. Thomas Khun, in his book The Structure of Scientific Revolutions (Kuhn, 1962), studied this phenomenon in depth. He describes the usual development pattern of modern science as one of successive transitions from one paradigm to another via scientific revolutions. Consequently, he associates this cyclical pattern of disruption with all intellectual advances thereafter, including Newton. Previously, from antiquity onwards and up to the 17th century, he argues that such a dynamic did not exist: schools of thought coexisted and derived strength from their association with a particular set of phenomena that their own theory could explain better than the others. Most importantly, scholars would take no belief for granted and would rebuild their field anew from its foundations: a pattern, Khun argues, that is incompatible with significant discovery and invention as scientific knowledge does not accrete. In the context of the previously mentioned suspension bridge, it would be akin to rediscovering from scratch, every time a suspension bridge is built, the basic mathematical principles of solid mechanics, such as Young’s elastic modulus or Hooke’s law.

Khun argues that these inefficiencies triggered the evolution of the modern, institutionalised, process of scientific discovery. Formal definitions of scientific groups meant that an individual no longer need to rebuild the respective field’s
foundations but could build upon a previous body of knowledge. Consequently, the generally addressed literature which was prevalent in previous centuries, such as Darwin’s *On the Origin of Species*, slowly evolves into specialised literature that is legible only to the respective specialists, or initiates of that respective group. Scientific differentiation, i.e., the emergence of new scientific groups centred around a new paradigm, happens by virtue of what Khun describes as “novelties”. Novelties are discoveries or inventions inadvertently produced by “a game” (scientific domain) played under one set of rules (paradigm) that, in order to be assimilated, require the elaboration of another set. For example, classical physics, originally one scientific domain, speciated into several distinct areas of research by the 20th century, namely mechanics, acoustics, optics and electromagnetism. In the contemporary design process, this scientific specialisation can be clearly seen in the composition of a team for a building project: alongside the architect there are structural, mechanical, electrical and civil engineers, in turn supported by specialists from space planning, urban design, landscape design as well as geotechnical engineering domains.

### 2.2.2 Professions and Conflict

Design problems are quintessentially cross-disciplinary problems that require expert knowledge from more domains than one individual can hope to master. Furthermore, the diversity of the stakeholders involved does not stem from just industrial (division of labour) or scientific (differentiation of science) groups: given the impact of a design project on its surroundings, a social and cultural dimension may well be present. Communication across these professional boundaries implies the integration of a wide range of diverse ontologies with their matching set of values. For example, from a design point of view, the choice of a specific material is maybe extremely positive if it will age well and fit cohesively with the surroundings. Nevertheless, from an engineering point of view, this has the potential to create additional problems that reflect negatively on the project in terms of environmental performance and energy usage. A certain functional distribution—more commercial space, and less housing—is a positive factor for the client, nevertheless has negative implications for the surrounding area. In these simplified binary examples, both parties are right, nevertheless their ontological definitions of value do not match. In the case of the latter example, the designer, not having access to the financial language used by the developer, cannot demonstrate the viability of his solution, and neither can the developer, not mastering the values of design, reach a compromise with the designer.

Further complicating the interaction of a stakeholder group is the fact that their professions—considered here as an abstract system—are not just a positive force in
society. Professions exercise control over their respective domains not only out of a selfless need for better standards, quality and precision, but as well from a selfish need to preserve their identity, expand and solidify their claim of expertise over a certain aspect of the material world (Larson, 2017, 1979). Andrew Abbott, in his book, *The Systems of Professions* (Abbott, 1992), takes this argument further and argues that professions do not operate independently of each other, but exist in a broader "ecological system". Professions, and even more so professional firms, continuously compete with each other over their ecological niches and seek to enlarge their jurisdictions and desire other domain's skills. This competitive dynamic is visible in the relationship between architects and structural engineers, especially in the case of high-profile projects with a high degree of geometric complexity. Projects such as bridges, airports, football stadiums, etc. require large amounts of ingenuity to both design and build. Subsequently, both the architect and engineer passively claim ownership over the creativity and importance of their roles in the design process by emphasising their contribution. This sometimes leads to heated debate: following a presentation regarding the collaboration process for the V&A Dundee centre between the engineering (Arup) and design team (Kengo Kuma Architects), done by a member of the engineering team (Clark, 2018), the discussion afterwards focused on the actual distribution of creative credit between the two disciplines. The presentation focused on a digitally enabled workflow envisioned by the engineering team which allowed the design team to modify project parameters in such a way that would produce valid engineering analysis and fabrication models, yet still satisfy the architects' need for flexibility. As follows, members of the audience from an architectural background were arguing that it was their profession that was the key behind the project's success and beauty, and that the engineers were just "problem solvers"; this approach of constraining the design team's input was penned as artificially limiting. Needless to say, the opposite was asserted by the members from the engineering disciplines by saying that it was their problem formulation and solutions which made the project possible in the first place.

This leads us to two observations, the first of which reinforces Abbott's ecological model of professions: as seen in the example above, professions seek to expand and defend their jurisdictions, recognition and leadership positions. Engineering expands into architectural design; corollary examples, of architectural design expanding into engineering are also widespread. The second observation is that collaboration between stakeholders facing a design assignment is not a given: the existence of a common problem to solve does not mean the persons involved in its resolution will start from a position of mutual trust. In other words, the disparate ontologies that societal differentiation gave birth to serve as a defensive mechanism to the
professional bodies that they belong to. Therefore, creating bridges between them for collaboration purposes—for example, between the engineering and the design team in the example above—can be seen as an offensive act by one party as it can dilute their internal ontological system and set of values. Subsequently, this leads to a deterioration of trust between stakeholders, as opposed to an enhancement: one of the paradoxes of modernity that will be discussed at length in the following section. This aspect has an important bearing on the formalisation of the alternative approaches used to investigate data representation in Chapter 4 (rather than imposing an ontology, allow end-users to evolve their own on a need-by basis), as well as classification in Chapter 5 (allow end-user to curate the data being shared rather than be forced to expose all of it).

2.2.3 Modernity, Transparency and Trust

The increasing specialisation and differentiation of the contemporary context has, in the scope of the design process, a dual nature. While benefiting from the increasing precision of knowledge, detail and quality from each of the individual domains, acts of design show an increasing difficulty in integrating all these domains due to their ontological drift away from each other. To shed light on the systems behind the process of this integration and reveal the underlying mechanisms of differentiation and cross-domain exchanges, this review now turns to Giddens’ institutional analysis of modernity.

In his book, *The Consequences of Modernity*, Giddens proposes disembedding as the underlying mechanism which allows differentiation, or functional specialisation, to happen (Giddens, 1991). Disembedding is understood as "the lifting out of social relations from local contexts of interaction and their restructuring across indefinite spans of time-space", and he identifies two principal manifestations of it: symbolic tokens and expert systems1 (Giddens, 1991, p. 22). The former is understood as an interchange medium that can be communicated or transacted without regard to its specific characteristics or context. Giddens gives the example of money as a symbolic token: "today, money proper is independent of the means whereby it is represented, taking the form of pure information" (Giddens, 1991, p. 25). As follows, symbolic tokens embody information that can be trusted to represent a specific fact, action or object across both a temporal dimension as well as a spatial one. In the case of the design process, for example, the architect’s building plans are symbolic tokens by

---

1 The term “expert-systems”, in this context, does not refer to its modern use as a piece of AI software, but to a cohesive system of professional, scientific and technical parts. Giddens’ full description follows in the main text.
which design intent is communicated, as are sections, elevations, three dimensional models, structural specifications and bills of materials: in lieu of “the real thing”, they are used to communicate information pertaining to reality. The latter disembedding mechanism, namely that of expert systems, is described as "systems of technical accomplishment or professional expertise that organise large areas of the material and social environment in which we live today" (Giddens, 1991, p. 68). Expert systems provide guarantees of expectations across time and space. For example, architecture is an expert system that guarantees design knowledge and the ability to envision a building; medicine guarantees expert care and treatment in the face of illness; a driver's license guarantees that one can safely operate a car; etc. Summing up, symbolic tokens provide informational, or factual guarantees by which individuals can communicate, whereas expert systems provide guarantees on knowledge and domain expertise.

In a similar manner to Khun, Giddens notes that expert systems develop their own proprietary language, values and tools that are often incomprehensible—opaque—to the outside public. He then argues that the crucial element for the functioning of a modern society that is based on transactions of symbolic tokens and expert systems is that of trust: trust is "confidence in the reliability of a person or system regarding a given set of outcomes or events"; trust is vested "in the correctness of abstract principles and technical knowledge" (Giddens, 1991, p. 88). For example, a person crossing a bridge inadvertently trusts the designers and the builders of the bridge to ensure that it will not collapse; an architect trusts the structural engineer to have correctly specified the building’s columns; a plane manufacturer trusts his parts supplier for material integrity of the delivered pieces; a person travelling by plane trusts the plane manufacturer, as well as the governmental bodies that oversee technical repairs, etc. Summing up, according to Giddens, trust, or confidence in abstract capacities, underpins all of modern’s society operations and transactions.

As follows, the design process relies on a large amount of trust between the technical and non-technical stakeholders involved in order to be productive and achieve its aims of, for example, delivering a building. In his critical paper, *The Tyranny of Light*, Tsoukas argues that trust needs to be continuously produced in such a system in order for it to function (Tsoukas, 1997).

It is not just the design process, but all late modern societies’ operations that are dependent on trustworthy knowledge for their functioning. In the contemporary context, knowledge, according to both Giddens and Tsoukas, is reduced to pure information, i.e., "objectified, commodified, abstract and decontextualized representations". This serves today’s modern informational society’s needs of
producing trust; knowledge, once transmuted to information, can be openly communicated and therefore lead to, or legitimise claims of, transparency and, ultimately, trust.

Nevertheless, Tsoukas reveals, this assumes a view of information ("a collection of free standing items") as being neutral and without room for interpretation; furthermore, its status is that of "an objective, thing-like entity" that "exists independently of human agents" (Tsoukas, 1997). Communication, in this scenario, is coerced into a purely technical model—as originally defined by Shannon and Weaver (1948)—that dehumanises it. In turn, this gives birth to one of the major problems in generating trust and understanding between persons and domains of knowledge: by stripping away the inferential dimension of communication, namely the human act of interpretation, information is generated in a way that stops being able to represent reality in a meaningful way; it diminishes one’s capacity for understanding and stops being useful as a transactional entity between stakeholders. For example, a building cannot be reduced to the sum of its parts. By deconstructing a building into its objective, quantifiable composing elements, namely walls, columns, windows, floor slabs, dimensions and proportions, etc., its informational representation will inadvertently lose aspects that the system cannot easily measure. These can be, for example, the building’s historical and cultural value, the way it engages with its surroundings, its personal value for its inhabitants or users, its symbolic value as a landmark, etc. This is equally valid for any act of communication, regardless of what it is designed to transmit: if a system would be designed to just measure the cultural aspects of a specific building, it would miss its potentially life-endangering engineering and structural flaws. Consequently, communication and information have a very strong, often overlooked, political dimension: information exists and is created for a purpose by an author, and if this intention is not easily inferred or made explicit, more information will have the effect of reducing transparency rather than increasing it (Tsoukas, 1997).

The ideal of transparency based on objectified knowledge hinders the building of trust between stakeholders due to the separation of information and intent, or purpose (Tsoukas, 1997). Furthermore, this dynamic undermines the trust that is necessary for expert systems to function correctly: because information is decontextualized, it is required that it is thereafter placed into a context where it can be interpreted. Nevertheless, these contexts differ and thus lead to different and potentially conflicting interpretations. Therefore, Tsoukas argues, trust is less likely to be achieved when more information on an expert system's inner workings is made available. A simplified anecdotal example can be formulated as an interaction between an architect and a developer: based on initial design input, the developer
creates a financial summary that he finds is too expensive. He then demands a
different solution from the architect, who then, due to lack of trust, questions the
financial calculations of the developer and attempts to provide his own, in a way that
would justify his design. Not being fully aware of the internal model of costing that
the developer uses, the architect’s version falls short when evaluated by the
developer. Thereafter, the developer, goes on to request specific design changes as
his trust in the architect’s design capabilities (projected from his suspicion on the
inability to correctly cost the project) is presently reduced. Needless to say, this cycle
repeats itself several times, increasing transparency. Nevertheless, in most cases, due
to the unresolvable ontological differences between the architect’s point of view and
that of the developer, it does not reach a successful outcome. This spiralling effect of
mistrust—the more transparent an expert system becomes, the less it will be trusted;
the less trusted an expert system is, the greater the calls for transparency—actively
undermines communication and the effective interaction of practitioners from
various domains.

Summing up, so far, the theories of Kuhn, Abbott, Giddens and Tsoukas have been
presented in an attempt to sketch a higher level theoretical framework of the
mechanisms and dynamics underpinning the rich interactions between actors. In
doing so, it is revealed that "differentiation"—the process by which new disciplines
and their representatives emerge—is (a) a product of both industrial and
manufacturing needs, through "division of labour" as theorised by Adam Smith,
Kant, Durkheim and others, as well as (b) scientific speciation as described by
Thomas Kuhn. Both differentiation mechanisms come with two very similar caveats,
namely the alienation of workers in the case of the division of labour (Marx) and the
increasing difficulty of a specialised scientific domain to interact with others outside
it (Kuhn) which are echoed by Tsoukas and Giddens. Subsequently, Giddens’
analysis of modern society explains the dynamic of societal differentiation and their
interaction through two related concepts. The first, expert systems, describes how
large areas of the material world are organised and shaped by specialised groups of
knowledge and expertise with proprietary systems of value and categorisation. The
second, symbolic tokens, explains how these systems communicate and exchange
information. For both of these systems to function correctly and interact with each
other, the former requires trust to be vested in abstract capacities, whereas in the case
of the latter, trust is vested in abstract information. Consequently, trust becomes a
central resource for the operation and interaction of differentiated and specialised
groups of disciplines and their represented stakeholders. As such, the friction from
within the design process can be explained as a lack of trust. Tsoukas argues that this
is due to the fact that modern society is based on a technical model of communication,
one in which information is dehumanised and stripped of its social and political
dimension. Information, once decontextualised, needs to be re-embedded in order to
be interpreted. When crossing domains of specialisation, Tsoukas argues,
information is interpreted in a different ontological framework and thus creates
conflicting interpretations through which the trust necessary for fruitful interactions
is undermined.

2.3 Design & Convergence

Specialisation constructs a productive environment in which specific phenomena
may be studied in more depth and thus contribute to a greater understanding of the
surrounding world. Where this dynamic of the modern society can be described as a
divergent process, design is the opposite: the act of creating a building—or shaping
any small part of the built environment—necessitates the coordination of a large and
diverse group of professions and stakeholders, or in other words, the convergence
and collaboration of multiple expert systems. Surprisingly, the same linguistic and
psychological mechanisms that underpin the creation of meaning—metaphors and
analogies—play both a divergent role by supporting modern society’s differentiation
and specialisation, as well as a convergent one by enabling a shared understanding
between different ontologies.

2.3.1 Metaphors and Conceptual Displacement

Metaphors play a key role in the emergence of new theories and domains, in science
and elsewhere. Building on Thomas Kuhn’s contemporary work on the dynamics of
scientific revolution, Donald Schön argues that metaphors and analogies, when used
in a projective way (as opposed to purely descriptive), lead to a process of conceptual
displacement (Schön, 2011). Similarly, Lakoff and Johnson argue that conceptual
metaphors—the usage of one idea in the terms of another—are used by people to
understand abstract concepts (1980).

Metaphors transcend their purely ornamental role and become the main vehicle for
the generation of new knowledge and, as such, underpin the linguistic and
psychological process of creativity; conceptual displacement can be seen as a process
by which old concepts or theories are fit to new situations. As follows, Schön argues
that this is an action of transmutation of ideas and concepts, and not just a purely
superficial transformation. Old and new concepts are simultaneously defined
through a sequence of transposition, interpretation and correction (Schön, 2011, p.
57). Subsequently, two functions of the metaphor emerge: a radical one, which lies at
the base of new knowledge, and a conservative one, which explain how old theories
underlie new ones. The former, radical function, can be illustrated by looking at the “sleeping metaphors” (Schön, 2011, p. 79) present in scientific language, e.g. “atomic wind”, “biological transducer”, “computer memory”. These phrases are indicative of concepts that have been carried over and act as projective models from one discipline to another. The later, conservative function of metaphors, is visible by fact that “Greek theories of scale, of atomic processes, and of vision […] have managed to preserve themselves virtually intact” until today (Schön, 2011, p. 139).

Sergio Sismondo, in his analysis of metaphorical thinking in scientific theories, argues that science itself is impossible without it “because metaphors are needed to provide the cognitive resources to explore new domains” (Sismondo, 1997, p. 127). In his view, metaphors take structure from one domain and apply it to another (Sismondo, 1997, p. 137). This act of conceptual reorganisation has a performative aspect by which a certain structure can be forced on a subject: “metaphors change some of the structure of associations of the vehicle, not just the tenor” (Sismondo, 1997, p. 141). He sums up by asserting that metaphors are asymmetrical descriptors that reflect concepts from one field to another, and by doing so, create new meanings and shared values. This process is illustrated above, in Figure 1.

Figure 1: Asymmetric reflections of meaning (left); The emergence of a new domain (right).

---

2 For an exhaustive study on metaphors and analogies in architecture, see Philip Steadman’s book *The Evolution of Designs* from 1979. Sadly beyond the scope of the current investigation, it traces a series of metaphors as they traversed the architectural realm, namely organic, biological, ecological, etc.

3 The tenor in a metaphor is the subject to which attributes are ascribed, and the vehicle is the object whose attributes are borrowed. For example, in the metaphor “argument is war”, “argument” is the tenor, whereas “war” is the vehicle.

---

34
Thus, Schön's conceptual displacement and Sismondo's asymmetric descriptors envision a dynamic process of divergence and speciation of new domains and their separate ontologies. Nevertheless, this process can be seen also from the perspective of the newly formed expert system—professional, scientific, or industrial specialisation—as not one of speciation, but as one of convergence by which the interaction between two different domains, having been productive, gave birth to a new, internally coherent ontology. In some cases, the original domains continue to exist independently of each other, with the new domain claiming its own identity. In other cases, the precursors may partially have folded in the new specialisation or ceased to exist. Kuhn noted that specialisation tends to widen the gulf between domains, and results in a communication problem between separate bodies of knowledge—ontologies are not necessarily interchangeable or, if evolving for a long time, legible by persons outside the respective field. Notwithstanding, it is important to assert that the process of differentiation is countered by convergence, and that both are based on the same linguistic and psychological mechanisms. Therefore, modern society is not described by biological phylogeny—an ever-branching tree of species and subspecies, but by one of cultural phylogeny, with the main difference being that domains do not speciate ad infinitum, but also merge into strands that may preserve the identity of the predecessors in a latticed pattern (Figure 2).

Subsequently, a design problem acts as a convergent point in this network of knowledge strands, attracting many domains and their representative stakeholders due to its complexity. Nevertheless, this network is not fully prescribed by the problem at the beginning, because, as shall be seen, the problem itself is not yet fully defined. In what follows, this investigation will look at how the mechanisms of differentiation and convergence, up to now described from a theoretical perspective, perform and affect the interactions of stakeholders in an applied design context.
2.3.2 Wicked Problems

Horst Rittel, together with Melvin Webber, changed the field of design by linking it with politics. They achieved this by studying planning (and design) problems from the point of view of a pluralist society that is dominated by complex and potentially confrontational interactions between stakeholders. In their paper, *Dilemmas in a General Theory of Planning*, Rittel and Webber distinguish between "benign" or "tame" problems and "wicked" problems (Rittel and Webber, 1973). Tame problems are well defined problems and have findable solutions that may easily be evaluated on a quantifiable scale. For example, a mathematical problem can be considered tame—it has a verifiable solution and one can know when the problem is solved; similarly, the sizing of a specific column based upon the load it carries, or defining the number of sanitary spaces needed based on building occupancy. On the other hand, wicked problems are problems that stem from the inner tension embedded in the contemporary modern society and its accompanying paradoxes: they are complex assignments that cannot be solved through a linear approach in the manner of tame problems.

One of the defining characteristics of a wicked problem is that its understanding is directly dependent on its proposed solution: “the information needed to understand the problem depends on one’s idea of solving it” (Rittel and Webber, 1973). Following this, wicked problems do not have a definitive stopping rule: each proposed solution generates a new formulation of the problem, thus leading to a new solution, ad infinitum. Rittel and Webber affirm that wicked problems are never solved because an optimal solution is found, but rather the process of solving them is halted due to external considerations, like running out of time, money or patience. In the case of the conflicting discussion regarding the jurisdiction of the architect’s and that of the engineer’s professions, when analysing the design process as a wicked problem, the success (or failure) of a project is determined by the iterative act of defining and solving the problem. For example, a virtuous cycle can be seen as (1) the architect defines the problem by proposing a design solution, (2) subsequently the engineers solve it, but in so doing contributes to a greater understanding of the problem, therefore (3) the architect redefines the problem by proposing a new design solution, at which point (4) the engineer comes up with a refined structural solution, and so on until a satisfactory solution is reached. As such, design solutions can be said to emerge gradually “as a product of incessant judgement subjected to critical

---

4 The linear approach is a first order system thinking approach. It also known as the waterfall model and consists of (1) collecting and analysing data, (2) formulating a solution and (3) implementing the solution.
argument” (Rittel and Webber, 1973) that has the effect of slowly evolving a shared set of values between the stakeholders involved.

Nevertheless, Rittel and Weber note that “diverse values are held by different groups of individuals—that what satisfies one may be abhorrent to another, that what comprises problem-solution for one is problem-generation for another” (Rittel and Webber, 1973). As observed by both Kuhn and Tsoukas, the differentiated and specialised modern society does not have one shared ontology; this needs to be constructed on a problem by problem basis between the interactional network of stakeholders that participate in its solving. This status of ontological uncertainty is, according to Hanna, an inherent feature of the creative design process. He describes it as an incertitude of how the “design problem or situation can be represented or modelled in the first place” (Hanna, 2014). As an example, he describes the process undertaken for the fabrication of a series of sculptures by Anthony Gormley during which the “design progressed by virtue of repeated changes in abstract representation” (Figure 3). These repeated alterations crossed the boundaries between several abstract and material domains, and each crossing redefined, or enriched, the problem definition. As follows, the design process contains more variables than can be represented in a finite model (Schön, 1991) or known from the onset, thus actions taken within it tend to produce consequences that cannot be predicted: an attempt to solve a problem will actually modify or change its representation, or definition.

### 2.3.3 Solving Wicked Problems

The act of ontological revision that underpins creative design, depending on the scale or breadth of the issue addressed, is performed either by one sole practitioner or within a stakeholder network. If the problem is within the remit of a single discipline, for example architecture, the process benefits from an internally coherent ontology,
or at least a set of matching or relatable concepts. But when the scope of the problem expands as such that more disciplines are involved—as is usual with most design problems—new dynamics of interpersonal and inter-professional communication come into play, and the problem being addressed becomes even more wicked.

Problem wickedness, together with social and technical complexity comprise the three main categories of forces that tend to fragment a project or “pull it apart”, according to Jeff Conklin (2005). Whereas they could be seen as symptoms of wickedness, he draws a distinction between them: “wickedness is described as a property of the problem/solution space and the cognitive dynamics of exploring that space, social complexity is a problem of the social network that is engaging with the problem” (ibid, p. 12). The wickedness of a problem is dangerous if the problem is misidentified as being “tame” or it is tamed in an unproductive way. Social complexity, on the other hand, arises from the structural relationships between stakeholders and often takes the form of “cultural wars” between departments or specialisations. Finally, technical complexity is potentially a fragmenting force, but not always. It arises from the large amounts of available technological solutions for a given problem, as well as the unintended side-effects of their interaction which cannot be fully predicted.

Conklin proposes that, in order for design problems to be resolved, organisations and the groups involved must act in such a way that demonstrates collective intelligence, an emergent form of collaboration that amplifies knowledge and expertise from one domain when it interacts with another—as opposed to, in other words, simple collaboration. Seen within this framework, simple collaboration can be considered as purely zero-sum game of informational exchange between parties that does not lead to new insights. Collective intelligence, on the other hand, is seen as a phenomenon in which “the whole is greater than the sum of the parts”.

It is therefore necessary that there is a sense of shared understanding as well as shared commitment in between the stakeholders involved in process. These two, when present simultaneously, amount to coherence, which according to Conklin, is “the antidote for fragmentation”. Defragmenting a project, as such, becomes an act of ensuring the existence of shared understanding of the project’s background and issues, of a shared commitment towards the project’s objectives and measures of success, and most importantly, of shared meaning for key terms and concepts amongst the stakeholders involved in the process (Bechky, 2003; Conklin, 2005; Lewicki et al., 1998; Walz et al., 1993).

In a manufacturing context, Bechky’s study on creating shared understanding reveals the importance of embedding the communication between the various stakeholders
in their respective work place contexts (Bechky, 2003). According to Bechky, understanding is “situational, cultural and contextual” therefore its creation process cannot rely on simple knowledge transfer. Echoing Giddens’ and Tsoukas’ concerns regarding disembodied (decontextualized) information, Bechky argues that communication problems arise between members of different groups due to a lack of common ground that leads to information being interpreted in different ways—reflecting the context in which it is received. Nevertheless, she illustrates through a case study in the manufacturing of processor units that, if members of these different groups start providing solutions (or defining the problems) based on the differences between their work contexts, a shared set of common meanings and terms arise that contribute to a richer understanding of the problem being faced. This is achieved sequentially, first through a set of transformative actions by which one stakeholder tries to conceptualise the knowledge of another within the context of his work. Gradually, these transformative actions accumulate and start shaping a common ground between the stakeholders (Bechky, 2003, p.234).

For example, the interaction between an architect and structural engineer involves many exchanges of information. Starting from one instance of a design, a structural solution is proposed. Nevertheless, this is not satisfactory for the architect: imagine, for example, a large overhanging body has the (previously unknown) effect of requiring an extra set of columns that are visible on the ground floor. This leads the architect to redefine the design with a much smaller overhang but a larger ground floor space as well, in order to hide, or incorporate the new columns. The structural engineer comes back with a revised solution that, given the smaller dimensions of the overhang, does not necessitate the columns that were initially the source of the architect’s irks. In an ideal scenario, the architect would meet with the structural engineer and contextualise their values in each other’s domain: the architect would soon gain a knowledge that overhangs above a certain length will require more supporting columns which potentially ruin the outside space along the building; the structural engineer would understand that a free sheltered public space is an important feature for the building in the eyes of the architect, and thus be able to inform his proposed solutions with this in mind. Essentially, “by exposing members of different communities [domains] to the perspectives and work of the others, these interactions reduce the differences in the understanding of the groups” (Bechky, 2003, p.236) and create a set of relationships between the ontologies involved in the project’s resolution.

It is important to note that the communication and collaboration context does not need to be fully cooperative—or the stakeholders do not need to trust each other blindly—for it to operate successfully and allow for “collective intelligence” to
emerge. Based on an analysis of communication inside a software practice, Walz et al. (1993) show that conflict facilitates the learning process that leads to shared understanding. In scripted social dialogue, conflict usually takes the form of either someone playing the role of the devil’s advocate, helping the decision maker test his assumptions, or as a dialectic scenario in which two sides exist that pit thesis and antithesis together. The latter approach allows, according to Walz et al., to generate the necessary information for decision making and clarify situations where there is more than way to define a problem (ibid.). Underpinning this view of productive conflict is Lewicki’s later research which shows that trust and distrust do not exclude each other and that their interaction can be productive (Lewicki et al., 1998). This is based on the finding that trust and distrust are not polar opposites of the same scale, but they are “functional equivalents” (ibid.) . Both trust and distrust function to reduce the complexity of a given problem (social or technical) by assuming beneficial outcomes in the case of the former and injurious ones in the case of the latter. Therefore, a state of ambivalence—the coexistence of trust and distrust—is a key driver in the dynamics of resolving or defragmenting wicked problems and, to a certain extent, sets the stage for Conklin’s state of coherence.

According to Weber and Khademian (Weber and Khademian, 2008), networks are seen as the best way to govern and manage wicked problems. They are defined by enduring exchange relationships between organisations, people and groups of people. Public and private governing structures have evolved from a hierarchical model of organisation to a networked one, as the latter demonstrates several advantages over the former when it comes to accomplishing complex tasks. Networks can share scarce resources and achieve collective goals better; they demonstrate flexibility, efficiency; they can accumulate vital resources and power needed to carry out shared tasks; etc. Most importantly, networks “allow participants to accomplish something collectively that could not be accomplished individually” (ibid.). Further reinforcing this view, from an analytical angle, is Watts and Strogatz’s famous analysis of small world networks. They conclude that “dynamical systems with small-world coupling”—i.e., semi-random networks not unlike a group of people involved in the design process—show evidence of “enhanced signal-propagation speed, computational power, and synchronizability” (Watts and Strogatz, 1998). The exchange relationships underlying a network—regardless of its goal—are, essentially, informational flows. From this point of view, Weber stresses the importance of transactional requirements of such communication—i.e., not just the information itself, but how it is transmitted and what is done with it. He describes effective networks as those being able to successfully transfer knowledge, ensure and communicate its receipt and comprehension, as well as integrate it into new
knowledge. As will be discussed later on in this chapter, Weber’s analysis, coming from an administrative and management angle, describes a vital ingredient of an inferential communication model, namely a system of evaluating receipt and comprehension of informational exchanges.

Where Weber describes these networks as mainly composed of human actors, Dawes et al. describe them as socio-technical systems in which human, organisations and institutions coexist “in a mutually influential relationship with processes, practices, software and other information technologies” (Dawes et al., 2009). He goes on to argue that, for such networks to be effective, they exhibit a culture in which the sender of information, by default, questions the relevancy of the information he produces, and, if it has even a remote chance of it being so, he or she will share it: “a need to share network culture” (ibid.). Given the symbiotic relationship between technology and its application, Dawes warns that approaching networks from purely an IT perspective is problematic. Networks undergo a constant evolution; nevertheless, established patterns of exchange do emerge. As such, in order to provide for this process of definition, development processes of early formed stakeholder groups “need to emphasise early, open dialogue and examination of assumptions and expectations”—thus supporting Lewicki’s claim that an ambivalent state of trust and distrust is productive, and similarly Walz’s notion of scripted conflict.

Summing up, it is important to note that shared understanding amongst a varied group of stakeholders does not entail the notion of a shared ontology. While the latter scenario would be ideal, it is an unreachable goal—as Roberts (2000) puts it, “getting the whole system in one room” is an impossibility. This is because the definition of the “system” itself is in constant flux, and thus cannot be contained in a “room”. The socio-technical network that aggregates around a wicked problem is continuously evolving as people, ideas, software, and other actors (both human and non-human) continuously influence each other and their collective interpretation of the problem. The connections between a network’s actors, as well as the actual actors themselves, change and mutate in time. As follows, coherence within this context can only be achieved by challenging and questioning information and knowledge continuously through dialogue. By doing so, a certain interdependence is created amongst key concepts from different ontologically different groups. In some exceptional cases, these can crystallise in a new discipline or profession, but in most, day-to-day wicked problem design assignments, simply understanding the trade-offs involved between one group’s values and another’s is enough to engender a productive cycle of define-solve iterations required to manage or resolve wicked problems.
2.4 Design and Communication

Design problems necessitate a large and diverse number of stakeholders to participate in their resolution, thus making the above normative approaches extremely relevant for tackling communication issues in a design process. The RIBA Design Management Guide describes the contemporary context as an intense and pressured environment due to the increasing complexity in managing the production, coordination and integration of design in projects (Sinclair, 2014). This is due to conflicting situations emerging from increasing accuracy demands in timing and costing as well as managing design input throughout both time and areas of expertise, like the various contractors and sub-contractors involved in the building process as well as other specialist groups. It can be said that today, “an examination of professional diaries is likely to show that most architects spend more time interacting with other specialist consultants and with fellow architects, than working in isolation” (Lawson, 2005, p. 239). Subsequently, the following section, by building on literature that tackles specifically how design professions cooperate and work together, will attempt to reveal the causes behind the friction in the communication process.

In *The Reflective Practitioner*, Donald Schön analyses how disciplines and individuals interact with a given problem and build both personal experience as well as domain wide useful knowledge (Schön, 1991). In a chapter dedicated to architecture, describes the design as a “reflective conversation with the situation” (*ibid.*, p. 76). In his analysis of the design process of an architecture student, supervised by her tutor, he reveals a certain protocol that unfolds in sequential moves. Each is a “local experiment which contributes to the global experiment of reframing the problem” (*ibid.*, p. 94), and thus represents a dialogue with the situation itself. An action in one direction or the other reveals certain aspects of the problem—positive or negative—which in turn, by contributing to the architect’s understanding of the problem, help in the planning of the next move. This communicative act between the formulation of a design problem and its solution is the main task of the designer. Because it happens internally—either within the designer’s own thought process, or within the discipline itself—Schön calls this process an act of reflection in which information and knowledge is actively reflected between the surface of a problem definition and its potential suggested solution.

Further supporting this argument is Lawson’s assertion that designers negotiate between “the problem and the solution view”, where the former is understood as “needs, desires, wishes and requirements” and the latter is described in terms of “material, forms, systems and components” (Lawson, 2005, p. 272). Lawson describes
the design process as a conversation that aims to reconcile conflict between these two views, during which it is not just the solution—or design—that evolves, but as well the problem definition. This is a co-evolution model of the design process that explains it as “a series of solution states each evolving from the previous one, in parallel to a series of problem states again each evolving from the previous one” (Lawson, 2005, p. 272; Maher and Poon, 1996).

Nevertheless, the effectiveness of this process has limitations. The need for interaction with external stakeholders—actors that come from another discipline, or simply people that vested interests in a problem by virtue of the context they are in—means that internal decisions must be publicly justified and introduces a political element to the exchange. The process of internal reflection described previously does not lend itself easily to such an act of making it transparent, even in a curated and carefully chosen way. In his analysis of the discipline of planning, Schön describes the interaction between an urban planner and a developer during the initial phases of an ultimately abortive potential project. Both parties were engaging in a process of internal “conversations with the situation” but instead of choosing to transparently reveal information, they chose to steer the situation in a managed way and “withheld negative information, tested assumptions privately, and sought to maintain unilateral control over the other” (Schön, 1991, p. 269). In Schön’s example, this led to both parties making wrong assumptions regarding each other’s interests, and the situation ultimately evolved into the developer’s decision to not go forward with the redevelopment plan. This protectionist attitude towards one’s private resources of knowledge is further supported by Lawson’s affirmation that “many designers seem to widen rather than bridge the gap between themselves and others” (Lawson, 2005, p. 234). Going beyond the popular perceptions of a designer as a creative genius, at a professional level this behaviour can be ascribed to the competing nature of expert systems or disciplines and, equally important, to the need to avoid Tsoukas’ paradox of transparency mentioned earlier in this chapter: too much transparency, by allowing information and knowledge to be interpreted in a different context than that in which it was created, allows for diverging and often conflicting interpretations and thus undermines the trust it was intended to build.

Nevertheless, reinforcing Giddens’ thoughts on modern society, Schön asserts that trust is exactly what is needed for such a reflective approach to work across individuals, disciplines or professions. This can be partly achieved by full disclosure of information, as it produces a climate in which parties are more disposed to understand each other’s point of view (Schön, 1991, p. 352). Furthermore, actors involved in the design process should engage with each other in a reflective contract—one in which both parties renounce attempts at “win/lose games of
control” (Schön, 1991, p. 303), and openly admit uncertainty (the basis for the process of reflection-in-action) without misinterpreting it as weakness or incompetence. In short, there is a growing need for cooperative inquiry within the (potential) adversarial context of the design process: ontological revision, or conceptual displacement, if done publicly and openly, can be often stigmatised or have adverse effects if taken advantage of.

How does this map onto the broader landscape of AEC? The RIBA Design Management guide, mentioned earlier in the introduction of this section, underlines the importance of the project manager as a communicator and enabler of dialogue throughout the stages of the design process. Nevertheless, when it comes to information exchanges, the suggested approach becomes very specific outlining “what [information] should be produced and, more specifically, when it should be produced, in support of a collaborative approach” (Sinclair, 2014, p. 3). While acknowledging that each project is unique, and therefore have different requirements in terms of data and collaboration flows based on project specifics, constraints and client objectives, there is a very clear emphasis of exerting control over the whole process through “rigorous application of standards and protocols” (Fairhead, 2015, p. 89). Ironically, exerting control inhibits the evolution of the stakeholder network surrounding a design problem and, to a certain extent, discourages the acceptance of uncertainty as an integral part of design. Consequently, while the assumption that communication will and should happen in order to achieve shared understanding and facilitate the management of a wicked design problem can be made, contrary to current trends in AEC, the assumption on how it will happen, and amongst whom cannot be asserted: the design process does not have a fixed topology and constituency, but—as mentioned in the previous section—is a socio-technical system in constant flux, even throughout the duration of a single project.

There has been a wide range of studies, reports and research projects coming from construction regarding collaboration and communication. Andrew Dainty’s analysis, by informing theoretical aspects of communication and management with the realities of practice, describes a rich and problematic context. Chief among the challenges identified is that of the fact that the AEC industry operates on a project basis (Dainty et al., 2006, p. 21). In his book, Communication in Construction, a project is defined as “an endeavour in which human, financial and material resources are organised in a novel way to undertake a unique scope of work” (ibid., p. 34). Projects, within the AEC world, present a different set of problems than in other industries, such as car manufacturing or industrial design. Specifically, every construction project (and, by extension, design project) is unique, essentially executed only once, with no prior prototyping session that can test assumptions and validate ideas; a
building is only built once. To put it differently, a project is a unique combination of client objectives and budget, site constraints, technological solutions, design and stakeholders that collaborate towards its enablement. A further observation is, according to Dainty et al., the fact that “barriers to effective communication are complex and multifarious because of the number of actors which govern the success of the construction processes” (Dainty et al., 2006, p. 18). These actors, given the temporal and finite nature of projects, will collaborate only for short periods of time leading, in turn, to barriers in standardisation and formulation of specific protocols, as each project will tend to build its own network of informal communications. Nevertheless, this is not necessarily a negative aspect: Dainty points out the Building Industry Communications Research Project report from 1966, which suggests that informal systems and procedures “seem to produce more realistic phasing of decisions and more realistic flexibility in the face of what is described as the inevitable uncertainties in the construction process” (ibid., 2006, p. 33).

The Building Industry Communications Research Project (BICRP) was a major study in the UK undertaken by the Tavistock Institute of Human Relations that explored the coordination problems of the construction industry. It ran from 1963 to 1966 and, as Alan Wild describes, during its progress, it recorded an evolutionary understanding and definition of the problem of communication in construction (Wild, 2004). It describes the design and build process as being underpinned by a “network of technical and social relationships”. Initially, the design phase of a project was seen as more complex than construction: “post-war evolution of techniques and material changes in the technical, economic and social context and methods of finance had generated a mismatch of roles”. Construction was seen as having “a sequence of operations […] capable of being foreseen with reasonable accuracy” (ibid., 2004). Nevertheless, this understanding changed during the course of the project. Interestingly, Wild reveals that based on an assessment of the “Briefing and Design” stage, the researchers “dropped their assumption of relative stability in construction” and essentially accepted that there is no comprehensive model. In 1965, the steering committee of the BICRP defined a more abstract view of the building process, which invalidated the self-containment of the building to process. A difference was made between the “directive” functions, or stages, of a project—briefing, designing, costing and construction—^5—and the adaptive functions. Where the normative design stages were fixed, the adaptive ones were unbounded and included “variations, post-

---

5 These have remained, to a certain extent, unchanged up to now. The RIBA Plan of Work specifies a similar breakdown of a project’s stages.
contract drawings, error and crisis exploitation, re-design, on-site design” (Wild, 2004). The obvious differences between the normative approaches and the informal ones led to the call for the integration of sociology and organisational research within BICRP in order to study the “psycho-social” system that was now emerging as the dominant problem definition model with the group (ibid.). Unfortunately, the study stopped at this stage due to lack of funding, and due to a lack of consensus in what context to carry it on, it was never continued.

Dainty et al. caution against applying a too simplistic model of communication when studying such a phenomenon; to a certain extent, assuming that construction was composed of a predictable sequence of operations led to the BICRP’s turbulent evolution. It is argued that the analogy between a technical (or, as it is otherwise known, code) model of communication and human communication “belie the complex melee of factors that shape and impact on the way in which messages are transferred from the transmitter to the receiver” and “ignores the physical and social context in which people work” (Dainty et al., 2006, p. 56). Reinforcing both Schön’s call for an environment in which collaborative inquiry can coexist within a potentially adversarial context, as well as the previous body of literature from the study of wicked problems, communication is seen as route to trust and mutual understanding that cannot be explicitly reduced to a series of norms (ibid., p. 230) without losing its emergent properties. The questioning of which model is best suited to study the communication flows of the AEC industries is a recurrent avenue of research.

Similarly to Dainty et al., Emmit and Gorse challenge as well the popular view of communication as a purely technical act of sending and receiving information. They state that different models may be more or less appropriate for different aspects of the process (Emmitt and Gorse, 2003, p. 33), and counterpose to the simplicity of Shannon-Weaver’s theory of communication a set of other configurations that take into account the peculiarities of the AEC industry.

Within this context, digital technologies are often discussed as potential positive force that can enable a more efficient flow of information, nevertheless Dainty cautions that their usage easily undermines “the tenets of effective human communication and interaction” (Dainty et al., 2006, p. 39). Furthermore, digital technologies “can inhibit explanation of meaning between project participants” (Dainty et al., 2006, p. 118). Essentially, the digital medium imposes, due its nature, a certain technical paradigm on the process. As the purview of this research project is intimately tied with digital design communication, it is of importance to understand what different communication models actually entail outside the rigours of a code-based environment and try and extract a set of guiding principles by which, within the last
section of this chapter, the contemporary applications of digital design communication in AEC will be assessed.

2.4.1 Communication Models

Different communication models can be applied to different parts of the communication processes within the AEC industries. The goal of the following section is not to come up with a model that fits digital communication within the AEC industries, or a tailored definition of the term itself. As the previous sections outline, successful or effective communication in the context of a wicked problem—be it related to design, construction, planning, or any other domain—is essentially a psycho-social process that is partly enabled by technology. In part, its effectiveness depends on unquantifiable human factors and personality traits that cannot be necessarily fully quantified, or even qualified. In another part, it also depends on the applicability and efficiency of formalised—one could say managerial—processes, like the RIBA Plan of Work, that give it structure. Notwithstanding, it also depends on the technology (digital or otherwise) that one has access to in order to enact communication. Technology can be seen as an application of means to achieve ends (Heidegger, 1978), nevertheless the ends (or goals) are also fungible or malleable (Blitz, 2014). Therefore, when considering digital technologies that enable communication, they should viewed as active, rather than subservient or passive, actors that both serve and shape that which they were originally designed to enable. The same can be said of the models and theories that are used to understand and analyse communication inside the design process.

According to Sperber and Wilson, there are two main categories of communication models: code-based and inferential. Code models of communication have in common the fact that a communication encodes their message into a signal, which thereafter is decoded by the audience using an identical copy of the code (Wilson and Sperber, 2008, p.607). Inferential models, on the other hand, posit that a sender “provides evidence of her intention to convey a certain meaning”, which thereafter is inferred by the audience based on evidence that has been provided and that is available (i.e., context). In other words, the act of encoding and decoding is not purely deterministic—or mechanical. It is rather a potentially non-deterministic process which involves various contextual enrichment mechanisms. As follows, both Shannon’s archetypal code model of communication (which has evolved from the engineering requirements of modern technology), as well as inferential models of communication which aim to capture the social dynamics of communication will be analysed. This will allow for the necessary context for this research project to emerge. Its digitally-oriented and applied nature would, at first sight, warrant a purely
technical approach to communication. Nevertheless, providing an inferential approach more closely tied with the psycho-social aspects of communication will enable a much more relevant analysis.

2.4.2 Code model of communication

The Shannon-Weaver model of communication has been hugely influential and has been widely adopted in social sciences, biology, psychology, organisational research, etc. It introduced an integrated model of information exchange (Figure 4) that incorporates the concepts of information source, transmitter, receiver, signal, channel, noise, encoding, decoding, etc. into one unifying theory (Shannon, 1948).

![Shannon's diagram of the communication model](Shannon, 1948).

According to Shannon, a communication system consists of five main parts:

1. An **information source** which produces a message or sequence of messages to be communicated to the receiving terminal. […]
2. A **transmitter** which operates on the message in some way to produce a signal suitable for transmission over the channel. […]
3. The **channel** is merely the medium used to transmit the signal from transmitter to receiver. […]
4. The **receiver** ordinarily performs the inverse operation of that done by the transmitter, reconstructing the message from the signal.
5. The **destination** is the person (or thing) for whom the message is intended.” (Shannon, 1948)

It is important to note that, previous to 1948, communication was mostly an engineering-related discipline, and it concerned itself with the practical problems posed by the advent of the telegraph (1830), telephone (1876), wireless telegraph (1887), radio (1900), television (1925), etc. (Verdu, 1998). Shannon’s theory emerged from this highly technical background and its ground-breaking contribution⁶ was to

---

⁶ Succinctly, Shannon, assuming that the smallest possible unit of information is a bit (a one or a zero), demonstrates that it is always possible to encode information in such a way so as to transmit it at the maximum rate that the channel allows (as defined by its entropy). The
establish a scientific base on which a whole new field—information theory—was developed and consequently led to the development of the digital age.

Nevertheless, Shannon frames his theory clearly in an engineering and scientific context. He acknowledges that messages have meaning: “they [messages] refer to or are correlated according to some system with certain physical or conceptual entities” (Shannon, 1948, p. 379) but he clearly differentiates his work from this view: “these semantic aspects of communication are irrelevant to the engineering problem” (ibid., p. 380). In the book that presented Shannon’s theory to a general audience, Weaver underlines the fact that Shannon’s model is purely technical, as opposed to semantic or qualitative. Consequently, “information must not be equated with meaning” (Shannon and Weaver, 1963). Nevertheless, earlier in this chapter, it has been shown how Tsoukas, nearly five decades later, criticises modern society’s decontextualised approach to information. In turn, this can be interpreted as the consequence of dismissing Shannon’s communication theory context, against both his and Weaver’s arguments. The distancing of information from its natural semantic and social embedding can also be traced to BICRP’s evolving understanding of the problem within the construction industry: from an initially clear and “tame” problem dealing with linear information exchanges, it became an unquantifiable “psycho-social” system full of informal qualities. As Tsoukas would put it, in a social context, “disembodied” information is detrimental to the process of communication.

2.4.3 Inferential models of communication

Consequently, the appropriation of Shannon’s model in other domains may carry across the marginalisation of the semantic aspects of communication, to the potential detriment of the respective field. Howard Garfinkel, a contemporary of Shannon who is known for his role in establishing ethnomethodology and conversation analysis as scientific fields, developed a separate information and communication theory, that was, nevertheless, not published until 40 years later. He counterpoints Shannon’s and his precursor’s technical models with a social theory of information, one of its most important tenets being that, according to Rawls, “information is not only put to social uses, but becomes information in the first place through situated social practices” (Garfinkel and Rawls, 2006, p. 11). His prior critique of quantification is similar to the one of Tsoukas’ and Strathern’s, namely that reducing social order to general trends removes unquantifiable—qualitative and detailed—processes that are crucial to understand it in the first place. To Garfinkel, information is not something that is

second most important statement is that it is always possible, no matter the noise of a channel, to devise an encoding scheme that allows for error free transmission of information.
recalled from an abstract storage, but is something that is re-created out of the “resources of the available order of possibilities of experience, available sensory materials, actions, etc.” (ibid., p. 158).

Garfinkel posits that information, from a social point of view, accumulates over the course of an exchange. Thus, previously transmitted information continuously enriches the context of its current interpretation and facilitates its transformation into meaning. An important condition underpinning this is, according to Garfinkel, continuity: interactions are seen to have a given “time horizon” of succession, which, if surpassed, causes an interruption in the “stream of experience” and thus breaks the communicative process (Garfinkel and Rawls, 2006, p. 186). Memory is a set of operations by which a previous meaningful experience can be reproduced or represented (ibid., 2006, p. 159). This describes a picture of communication in which information processes and social interaction are interwoven. Consequently, Garfinkel sees communication not just as a deterministic exchange of symbols that Shannon’s model describes, but a cooperative process through which actors, through an ordered exchange of symbols, facilitate mutual intelligibility (ibid., p. 15). As such, it is important to note that the cooperative aspect of communication is an important notion: not restricted to Garfinkel’s work, it surfaces as well in Grice’s inferential model of communication, which shall be shortly brought forward. Moreover, it is inherently linked to the same concept present in the previous analysis of normative approaches to design and wicked problem resolution.

In 1975, Grice introduced the “Cooperative Principle” in his essay *Logic and Conversation*. It describes how receivers and senders of information must act cooperatively and mutually accept each other in order to be understood. In other words, “make your conversational contribution such as is required at the stage at which it occurs, by the accepted purpose or direction of talk exchange in which you are currently engaged” (Grice, 1991, p. 27) and according to Grice, any rational conversation must follow it. It comprises of four maxims, namely: (1) the maxim of quality, (2) the maxim of quantity, (3) the maxim of relevance, (4) the maxim of manner, which are summarised below:

“(1) *quantity*: make your contribution as informative as it is required; do not make your contribution more informative than it is required [...]  
(2) *quality*: make your contribution one that it is true [...]  
(3) *relation*: “be relevant” [...]  
(4) *manner*: avoid obscurity and ambiguity, be brief and orderly [...]”

(Grice, 1991, p. 26-28)
Grice notes that the cooperative principle does not apply only to verbal exchanges, but also to any cooperative transactions, such as those happening in the AEC industries. The maxim of quantity can be exemplified as: if a structural engineer is assigned with a load analysis of a building, he will most probably expect to receive not the full detailed 3D model, but the schematic engineering model. Similarly, the maxim of quality: the architect expects back real volumetric dimensions of the structural elements, not spurious ones. The maxim of relevance: the person in charge of quantifications of the project will expect to receive the gross dimensions for the building elements, and not the fire escape plan. The maxim of manner implies that participants expect of each other that they “execute their performance with reasonable dispatch” (Grice, 1991, p. 28).

Earlier in this chapter it has been shown that cooperation inside varied stakeholder networks does not necessarily imply a wholly non-adversarial context (one in which relationships are fully positive). Trust and mistrust can coexist and, furthermore, different disciplines, domains or actors may have conflicting definitions of value. Grice, when analysing the features of cooperative transactions, takes this fact into account and elegantly says that, while participants must have some common immediate aim, their ultimate aims may vary: they can be independent of each or even in conflict. In other words, an altruistic architect’s ultimate goal might be to design the best building as defined, for example, aesthetically; a developer’s ultimate goal might be to, for instance, maximise his profit. Even if these two actor’s goals are in conflict, they will engage in cooperative transactions in order to resolve—and define—sub-tasks that will progress their individual goals. Furthermore, Grice reinforces Garfinkel’s view of the importance of sequentiality (a linear succession of events) in communication. The second feature of cooperative transactions that he identifies is that the contributions of the participants should be mutually dependent, or “dovetailed” (Grice, 1991, p. 29). In essence, transactions in which information is shared, but not used—or does not become depended upon—are not cooperative. An actor that unilaterally broadcasts information blindly is adding “noise” to the system, as the information sent does not enrich the context in which it can be interpreted and thus prevents its transformation into meaning, as described by Garfinkel7, and, as such, is not relevant.

The issue of relevance becomes central in the work of Sperber and Wilson. By building on the inferential model of communication outlined by Grice, they expand

---

7 In digital communication, as will be discussed later on in this thesis, measuring relevance becomes a key requirement, as it is not directly evident to the sender whether the information they transmit is being used or not.
the third maxim—“be relevant”—and propose a complementary theory of communication, namely relevance theory. Its central claim is that “the expectations of relevance raised by an utterance are precise and predictable enough to guide the hearer towards the speaker’s meaning” (Wilson and Sperber, 2008, p. 607). This is based on the fact that when something is communicated, expectations are automatically created as to their quantity, quality, relation and manner. The receiver, based on these expectations, is thereafter inferring the meaning of the speaker’s communicated information. Sperber and Wilson, nevertheless, question the rigidity of Grice’s four maxims and propose an alternative theory informed by human cognition. They state that human cognition is geared towards the maximisation of relevance, and that this alone is sufficient to explain inferential communication (ibid., p. 609).

An input is relevant to an individual when it “connects with background information he has available to yield conclusions that matter to him” (ibid., p. 610). For example, the information that “there are delays on the Victoria line” (input) can lead to a complete reorganisation of an individual’s travel arrangements (context), as there would not be enough time to catch the subsequent connection: the individual would potentially even opt for a complete different means of public transport. Such a cognitive effect is described by Sperber and Wilson as being positive, as opposed to passive. The later, when contextualised, does not trigger any major actions or the generation of new knowledge (for example, “water is wet”). Conversely, the former enables a change in an individual’s representation of the world. Taking the example above, if the individual were not to need the Victoria line, they wouldn’t benefit from any positive cognitive effect; nevertheless, since they do need to get to King’s Cross and their current fastest option is taking the Victoria line from Vauxhall, the announcement of the delays prompt him to take the Northern line from nearby Oval. As such, the input did have a positive cognitive effect—the receiver’s understanding of the world changed.

Nevertheless, there is another factor to take into account when judging the relevance of an input, namely processing effort. The previous announcement regarding the delays on the Victoria line can also be formulated as “the following lines have good service: Metropolitan, Northern, Jubilee, Piccadilly, … [all besides the Victoria line]. The remaining lines have major delays”. Alternatively, “The line represented by the light blue colour has major delays”. In the case of the former reformulation, the processing of the information being communicated has a significantly higher cost, as it directly requires the receiver to listen to an iteration of all lines, and infer by omission if the one he is interested in has delays. Similarly, the latter requires a greater cognitive effort too: it requires to remember the association between the way
the tube line is represented on a map (light-blue) and its actual name (Victoria), a process which is not necessarily straightforward for someone who is not a local. All three statements are factually correct, nevertheless their relevance differs.

Sperber and Wilson posit that the relevance of an input is not a binary attribute (it is relevant, or it is not relevant), but rather a gradient. Accordingly, it can be assessed as follows: (1) all other things equal, the greater the positive cognitive effects achieved by processing an input, the greater the relevance of the input; (2) all other things equal, the greater the processing effort, the lower the relevance of the input (Wilson and Sperber, 2008, p. 609). The human cognitive system has evolved to maximise the relevance of an input in order to make the most of the available processing power. Consequently, Sperber and Wilson deduce that information will follow, or tends to follow, a path of least effort (ibid., p. 610). It is enriched at both an explicit and implicit level by the receiver as well as the sender.

Social groups follow, in their communicative transactions, this path of least effort by continuously maximising relevance and minimising cognitive processing. Becky’s account of the emergence of a “shared context” amongst stakeholders, documented earlier in this chapter, reinforces this dynamic view of communication as an act of continuous optimisation of informational content and its transformation into meaning. Essentially, this is a dynamic process through which the context of the transactions is continuously enriched by the information exchanged between the involved actors. The exchanged information, being reliant on this context, is also dependent on it for its encoding; processing effort is minimised by relying on the information enrichment resources thus being made available.

This allows us to partly explain why informal methods of communication, such as the ones described by the BICRP, as well as contemporary ones from the AEC industries, will always surface. Maximising relevance is a dynamic process, which conflicts with the static nature of a formalised approach.

Within the context of this applied research project, communication could have been framed as a purely technical endeavour of getting digital information from point A to point B in the most efficient way possible. Nevertheless, this would have led towards a barren view of the context in which the wider implications of societal embedding would have been ignored. For example, according to Rawls, the dissociation of meaning and information has come to cause difficulties back into technical disciplines, such as in system design: engineers found that “existing semantic—or schema based—approaches limit their effectiveness and that of their designs” because of the effort to re-forge a relationship between the data itself and its schema (Garfinkel and Rawls, 2006, p. 11). However, contemporary computer
science techniques have found ways to deal with this problem at a technical level, for example in the form of self-describing data structures.

Nevertheless, it is clear that information, especially within the communicative processes that lead to the resolution of design—and by extension, of wicked problems—is a societal accomplishment. It is based on a cooperative framework in which shared understanding and mutual legibility can emerge amongst the varied group of actors involved in its enactment. The following section shall review several digital communication technologies from the AEC industries to understand how they enable—or not—an act of collaborative communication by balancing the constraints coming from their inherently technical implementations with the requirements of a cooperative, socially embedded, dialogue.

2.5 State of The Art: Digital Communication & Design

Within the field of AEC, there has been a continuous research and development effort geared towards the development of software systems that, by leveraging the technological advances of both computers and their potential for connecting people (mainly based on the infrastructure provided by the internet network), would enhance communication within the design and construction process. Nevertheless, most recent discourse on digital collaboration has fallen under the monopoly of the term BIM, which provides a technically and an institutionally biased context in which to study the communicative processes of the industry (Dainty et al., 2017). This domain, nevertheless, has a far wider and richer history that dates back to the late 1960s and early 1970s and, to a certain extent, emerged from attempts to provide instruments for the management of wicked problems, and by extension, their domain of origin: design.

Unsurprisingly, given Rittel’s background in planning and design theory, an early information system to tackle the complexity of collaborative design and planning was first defined by him and Werner Kunz in the 1960s (Kunz et al., 1970). Named IBIS (Issue Based Information System), it assumed that the design process is one of argumentation that is centred around questions (issues) around which a series of plausible answers congregate. Around these, alternative positions (answers, either) are articulated by the participants in the process, until finally a consensus is reached, or the initial problem definition is reformulated (thus giving birth to a new set of issues). The goals of IBIS were to enable a higher degree of transparency and, consequently, help the decision-making process in design scenarios. By being implemented as a graph database (with nodes representing issues, positions and arguments, and edges the casual relationship between them), one could essentially
trace the history of the argument and its resolution and transformation, thus providing an informational instrument for structuring a design process (Kunz et al., 1970).

By building on top of the theoretical concepts of IBIS, R. McCall et al. propose a quasi-hierarchical, functionally structured database as the basis for a designer’s workstation (McCall et al., 1984). They justify the designer’s need for such an information system through a careful analysis of the nature of the design process itself, as well as of the data being produced and consumed within it. At a higher level, they state that the “collection of data […] is so large and the tasks so varied and changing” that, without a computer-based information system, the designer would not be able to do his job (McCall et al., 1984, p. 302.). In analysing the nature of data within the design process, the authors reveal two distinctions that separate it from “tame” data. The first is that it is usually produced as a consequence of itself or, in other words, it is generated by the design process itself in a response to previously generated data; it is not simply collected, or accumulated. This high internal dependency is due to the wicked nature of design problems which require constant cycles of re-evaluation of the assignment. The second distinction bears a striking reminder to the importance of context, and essentially draws attention to the fact that communication, within the design process, is inferential in nature: “design information is only information with respect to the individual projects in which it is used and with respect to those project sub-tasks for which it is useful” (ibid., 1984, p. 306).

Furthermore, they advocate for the openness and accessibility of such a system: “It must promote communication between the designer, his clients, and others who may be involved in the design process” (McCall et al., 1984). In such a system, traditional modelling or drafting would play only a part of many, while the overarching goal would be to “structure, and sustain” the discussion among the actors involved in the design process, a goal which is surreptitiously absent from most contemporary CAD software. Consequently, R. McCall et. al. stress the importance of seeing information more than just the raw design model data: “any data used to devise, find, or evaluate design information could also be design information” (ibid., 1984, p. 307).

An earlier example of using digital tools as a communication instrument that perfectly illustrates the potential of an accessible and integrative design system is Aish and Maver’s work on a collaborative design environment named PARTIAL (Aish and Fleming, 1977). PARTIAL is a parametric computer aided design system geared towards defining an environment in which both professional designers (architects) and end-users can collaboratively design and evaluate a particular
building type. The intention was to “provide a context where the designers/participants could combine their subjective design ideas with the necessary technical requirements” (Aish and Fleming, 1977). PARTIAL enabled all participants to “operate directly on [...] a complex, multivariate, but nevertheless, extremely real decision making process” through the use of digital design tools. Most importantly, it enabled design information to coexist with its evaluation, in a real time: as participants were drawing plans, it was presenting them with a summary evaluation of their design according to a certain set of criteria, such as surface area, connectivity, noise and cost.

Through subsequent design iterations, an end user of PARTIAL is made aware of what the authors call “fuzzy causalities” between the design itself and the various evaluation criteria. For example, a design that minimises hallways and thus scores high on connectivity does not perform well in terms of noise because it packs rooms too tightly to each other without regard to their function. The relationships between design geometry, objective performance measures, subjective choices and, in the end, the designer’s intuition represent a digital embodiment of the potentially adversarial context in which the design disciplines usually operate as described by Schön. As follows, Aish and Maver situate the use of computers in AEC as not only a productivity tool, but also as an “educational” instrument that allows wider engagement with the stakeholders involved in the design process which would “enable all participants to operate directly on, and hence learn about, a complex, multi-variate, fuzzy, but nevertheless real decision making process” (Aish and Fleming, 1977).

Relying on the advent of the World Wide Web and the increasing electronic connectivity it brought to people, prototypical tools were developed to study the process of information exchange in the design process. Previous research was embedded in the context of singular workstations that were expensive to operate and were not inter-connected, forcing a spatially bounded approach to design collaboration – participants needed to be in the same room, and/or have access to a computer. Nevertheless, between the 1970s and 1990s, computing became more and more accessible to businesses and people alike. De Vries studied the information exchange process in the building industry and proposed formalising it around a messaging protocol that captures the communicative network as it emerges around various actors and the tasks they execute (de Vries, 1995). He notes that this process crosses the organisational borders of a singular firm, and that, ultimately, “the organisational structure of the project controls the information flow between the participants” (ibid., p. 467). In an experiment (Figure 5), by tracking the messages exchanged between a simulated building project, he assembles an activity network
that shows the informational flow between activities:

![Activity network diagram (left); Message exchange scenario diagram (right) (de Vries, 1995, p. 472)](image)

The diagrams above are made possible by tracking the name of the sender, the recipient’s name, the activity’s name, the objects (geometric data) and the time when the message was sent. The author argues that this allows to understand which activities are interdependent, which ones can take place concurrently, potential bottlenecks as well as which actions need to be re-executed if the input data changes. Expanding beyond tracking the communicative flows between people, the author proposes that such a system can be used as a basis for automation of verification tasks: “verification can be highly automated if the message that is received by the design verification company could serve as input for another computer (FEM) application” (de Vries, 1995). Today, automation does not need to be restricted to purely engineering analysis; given the wide adoption of generative techniques throughout the AEC industry which can serve as the basis for automation, the application scope can be expanded to include more complex design tasks. Summing up, de Vries describes a prototypical digital system in which, on the basis of digital communicative actions, a process model of the design, analysis and construction activities can be created, monitored and shaped through automation. From a different point of view, one can describe it as a “parametric graph model” in which the nodes are not necessarily computational functions, but also human stakeholders.

In his paper, *Collaborative Design using Long Transactions and ‘Change Merge’*, Aish describes a set of computing principles and their relevance to “collaborative engineering across temporal, spatial and discipline dimensions” (Aish, 2000). Specifically, he proposes a novel way of parallelising design work by allowing end-users to independently work on subsets of a digital model and coordinate, at their own pace, by starting a “long transaction”. Coordination and synchronisation of the
whole is done through “change merge” operations, through which one designer reconciles his work with a central model (Figure 5). Essentially, this is an act of gathering the “parallel ‘strands’ back into a single unified and re- solved design statement” (Aish, 2000).

![Figure 6: Workflow diagram for Project Bank describing the process of branching and merging in a two-user scenario (Aish, 2000).](image)

The paper presents several key innovations in terms of computational techniques, such as the efficient usage of delta compression by storing only the differences between changes, rather than the whole model, which enable its proposed streamlined, loosely-coupled workflow between designers. Furthermore, Aish identifies several key requirements of the collaborative design process that can be seen as Human Computer Interaction (HCI) building block of digitally mediated design communication. Firstly, information sent must be curated by the user, because one “does not want to publish incomplete intermediate solutions, nor does he wish to be bombarded by with other users’ incomplete intermediate solutions” (Aish, 2000). Secondly, while discussing the process of merging, parallel modelling, users should get notified when someone else updates the repository. By doing so, one can be made aware of the evolution of the design data and subsequently update his local copy, thus ensuring that their modelling efforts are not being based on stale information (ibid.).

The importance of the design process is increasingly discussed in contemporary literature, and, as such, the workflows underpinning current design projects gain recognition as being key (and almost as playing equal parts) in the success of building projects. BIM originally evolved from intelligent parametric modelling systems, such as GLIDE (Eastman and Henrion, 1976), RUCAPS (Eastman et al., 2008) and Sonata,
that streamlined the drafting process of plans, sections and elevations from a 2.5D or 3D digital model of a building that comprised of linked object assemblies (i.e., a window would be contained by a wall; moving the wall would move the window as well). Nevertheless, over time it started to encompass methodologies on how the actors involved in the design process should collaborate and exchange information. BIM communicative strategies have as their basis two central aspects, namely a shared data model and information exchange workflows (Eastman et al., 2008).

A data model documents and specifies the nature and structure of a given domain’s digital information; in the case of the AEC industries, the accepted standard is IFC (Industry Foundation Classes). They comprise of more than 1000 elements that describe a rich variety of “product information”, i.e. the component elements of a building such as walls, doors, windows, as well as geometric elements, such as points, lines, polylines, curves, surfaces, etc. IFC’s ambition is to encompass all possible elements that describe the built environment in its digital form; consequently, by comparison to other digital standards that specify information structure, such as HTML (which comprises ±100 elements), IFC is rightfully perceived as being huge and difficult to comprehend.

The comparison between these two standards is, in most ways, invalid as they serve different purposes: one governs the built environment, the other structures web documents. Nevertheless, one difference that emerges from their juxtaposition is composability. While HTML is inherently modular, and designed around the assumption that one cannot fully encompass all the possible ways in which information will be structured within a web document, the approach of IFC is different: it aims to specify all the possible elements making up the built environment, essentially to be complete. Consequently, IFC, even though it is extensible to a certain extent, sacrifices the possibility of ontological revision at a representational level for the consistency that it brings, thus bringing in question its suitability for communicative purposes.

Most BIM communication workflows use IFC as an underlying information exchange format (van Berlo et al., 2012), or at least strongly mandate the use of a common shared one (Fairhead, 2015). On top of this shared data format, usually a central data repository is as well stipulated in order for it to play the role of the “one source of truth” for any aspects of a building project. All stakeholders involved in the design process are required to operate their information exchanges through this shared project model (ibid.). These information exchanges must follow specific protocols, which in the UK are governed by an official standard (Fairhead, 2015; Sinclair, 2014) that govern what information is produced, when it is produced, with whom it is
shared, etc. For example, the RIBA guidelines state that “properly defined information exchanges ensure that the right information is produced at the right time and so are a fundamental starting point for defining the overall scope of the project and the project team” (Fairhead, 2015) as well as clarifying legal and data ownership issues (Sinclair, 2014). What is surprising is the assumption that this approach entails the fact that one will know, at the beginning of a project, with whom one would communicate and exchange information throughout all the design stages. Nevertheless, as previously discussed, a design project is a quintessential wicked problem: ill-defined and with loose boundaries. Solving it, or more accurately put, managing it, will most probably require its redefinition and, consequently, a group of expertise and stakeholders that will change with time as well in its composition. Consequently, this suggests that BIM methodologies that deal with communication trade off, yet again, flexibility in exchange for more consistency and predictability of the design process.

Further evidence, albeit circumstantial, that existing “formal” digital methodologies do not sufficiently support design communication needs can be inferred from the proliferation of digital design tools, scripts and instruments that enable “informal” communication workflows. Most of these are project or company specific, and do not have the rigour that official BIM standards require, nevertheless they permeate the AEC industry. For example, Bhooshan describes the digital workflows utilised by Zaha Hadid Architects as hybrid, or a mixture between linear, manufacturing derived processes and non-linear, project specific collaborative workflows that essentially map complex geometrical surfaces to industry standard components, such as column and glazing systems (Bhoosan, 2017). While the specific complex geometric nature of Zaha Hadid Architect’s work can easily justify non-standard workflows, Deutsch describes a more wide-spread scenario where the design team needed to quickly iterate through design options and exchange data between two different software packages, namely Rhino and Revit: “after numerous one way workflows were explored, [the final solution relied on] exchanging information […] via comma-separated values (CSV) files” (Deutsch, 2017). Furthermore, this kind of practices, and many more similar approaches, have given birth to an entire panoply of custom software plugins that automate various aspects of the workflow. These were, in most part, written by expert users – engineers, architects, etc. – that had programming skills, and not by the established software vendors. A prominent example is van der Heijden’s Elefront plugin for Grasshopper that allowed designers to attach any kind of custom data to geometry objects, essentially allowing for users to create their own relational database inside Rhino. Flux.io, a Google funded start-up that, at the time of writing, ceased operations, provided an ecosystem of plugins
for the major design platforms (Revit, Rhino, SketchUp, etc.) that enabled designers and engineers to share model data in a real-time and ad-hoc manner, without enforcing any restrictions on the network topology.

In a recent article, Goldup et al. describe the nature of an internal platform for design data management and analysis that was developed for a large airport terminal project. At its core, it had a relational SQL database to which a number of other clients were connected through various application specific plugins, such as Slingshot for Grasshopper and Rhino, Excel, Revit, etc. Further interconnectivity was achieved through custom Python scripts that provided both read and write capabilities, but as well were performing structural analysis on the project data residing in the database (Goldup et al., 2017). Interestingly, the authors’ justification of the decision to develop custom software is rooted in the fact that, at the onset of the project, it was not clear what in what ways “the design data would be used, manipulated, shared and documented” (ibid., 2017). Consequently, “the design team chose to adopt a framework for storing and managing building information that was truly flexible: one that did not lock project data in a proprietary format, and ensured that it could be transplanted to other platforms quickly and easily as the right project workflow revealed itself” (ibid.). This pragmatic analysis of the status quo reinforces the fact that design projects are wicked, and that, in resolving and dealing with wickedness, one needs flexibility in order to be able to enact the communicative processes that lead to solution of the assignment.

Kvan’s provocative paper *Collaborative Design: What is it?* from 2000 dismisses “computer supported collaborative design” as an unenforceable goal (Kvan, 2000). He does this by distinguishing between collaboration and cooperation, arguing that the latter, rather than the former, is the norm within the design and build process. Companies and individuals do not collaborate – as collaboration, in its true form, requires “relationship building” and is time consuming – but mostly cooperate and compromise. As design actions are loosely coupled, digital tools “that support cooperative design should [also] be designed as loosely coupled components which allow for different systems and are reliant on different communication channels” (Kvan, 2000). Bart de Vries’ research into information exchange in the built environment was following this trajectory: it did not impose any strict top-down methodologies of how communication should be structured, or what software to be used, or what information should be exchanged and with what frequency; the stakeholder network was allowed to freely evolve its own strategy for exchanging information. Similarly, Robert Aish’s development work on Bentley’s Project Bank, even though restricted to one software ecosystem, was essentially enabling designers and engineers, through “long transactions”, to work in parallel on any subset of the
design model as they see fit. Seen from a different lens, Kvan’s statement can be re-interpreted as a call for embracing the richness and diversity of built environment industries and stakeholders and, subsequently, develop software systems that unobtrusively support and enhance them.

Summing up, Robert Aish notes that the “BIM should not be considered a philosophy of design” (Aish, 2014). Nevertheless, Green in the context of construction industry, comments on the fact that, nowadays, this is actually a reality: “[...] it seemingly doesn’t really matter what the problem is – the answer is always BIM” (Green, 2013, p. 12). According to Aish, the “BIM era” was the precursor of the “computational era”, whose objectives were to first overcome the limitations of the former in terms of automating design elements generation instead of manually creating them, and second to “move away from hardcoded building semantics and allow the designer to create his own components” (Aish, 2014, p. 42). Even though both eras flourish today, the regulatory nature of BIM and its drive for consistency and predictability has created an artificial rift between “formal” approaches and informal ones. Consequently, the aim of this research project can be partly formulated as an investigation into whether the demarcation between consistency and flexibility can be healed, and, furthermore, assess if it is actually justified in the contemporary technological context by attempting to challenge several assumptions of how the digital medium is used to communicate design data.
3. Methodology
The research presented in this thesis was undertaken within a collaborative network of academic and industrial partners. One of the main goals of this network, InnoChain, was to leverage the potential synergies between theoretical research, embedded within the academic partners, and practical concerns coming from practice. Consequently, the research process that underpins the theoretical investigations into digital design communication presented here has an important applied, or practical, component that was in constant dialogue with the conceptual aspects of this project.

3.1 Living Laboratory

From a methodological point of view, this research project has relied on the methods and strategies incorporated by a living laboratory. The concept of a living laboratory is attributed to William Mitchell, the dean of the School of Architecture and Planning from MIT, and it evolved out of the HCI and User Experience (UX) fields (Almirall and Wareham, 2011; Guzmán et al., 2013; Kusiak, 2007). Originally, it was used by Mitchell to study how users would interact with new technology artefacts starting to permeate buildings. Veli Pekka Niitamo described it as “a research methodology for sensing, prototyping, validating and refining complex solutions in multiple and evolving real-life contexts” (Almirall and Wareham, 2011).

More recent definitions of the term describe it as a “user centred innovation ecosystem integrating research and innovation” (Pallot, 2012). In a comparison with concurrent engineering, Guzmán et al. refer to living labs as research-based “infrastructure within which software companies and research organisations collaborate with lead users and early adopters to […] define, design, develop, and validate new products or services” (Guzmán et al., 2013). Across literature, there are two recurring methodological directives associated with a living lab: first, involve users as early as possible in the research process, and, second, experiment in real world settings with tangible artefacts (Guzmán et al., 2013; Pallot, 2012). The reasoning behind the former is that, by engaging all stakeholders from the beginning of the research process, emerging scenarios, usages and behaviours can be discovered, and help shape and validate the research problem. The latter stresses the importance of doing “research in the wild”, as opposed to within a laboratory, controlled environment using real, functioning products, rather than virtual mock-ups. The key difference between a prototype and a product (which, nevertheless, could be at a prototypical stage) is that a prototype, by itself, can only be used to simulate the accomplishment of a given task and thus implies a safe zone in which risk is eliminated. A working product, nevertheless, does not eliminate the risk involved in solving a specific assignment, and thus enables a more realistic
assessment. Simultaneously, it will allow for the discovery and investigation of aspects of the research problem which would otherwise have remained invisible in a low-risk, “staged” setting.

Within the field of HCI, there is a large body of studies analysing the affordances and disadvantages of moving research out of a controlled environment. One key advantage is that an “in-situ study is likely to reveal more of the kinds of problems and behaviours people will have and adopt”, thus gaining “ecological validity” (Rogers et al., 2017). Naturally, this comes with a potential loss of control over the experiments being performed, as there is no predefined set of tasks users must perform, and no control predictions over the assumed set of interactions (Rogers et al., 2017).

Nevertheless, the greater ecological validity of in the wild research, staged within a living laboratory context, is a needed complement to the current state of the art in digital design communication, as there is a growing divide between research and applications in BIM and “the small practitioner” (Dainty et al., 2017). The mismatch between the needs of the industry “on the ground” and the official policy mandated by national governments or large advisory bodies is amplified by the technological cost of implementing them. Consequently, this results in what Dainty calls a feeling of disenfranchisement across the AEC industry (Dainty et al., 2017). Notwithstanding, the problem of technological adoption is further reinforced by both the AEC industry’s resistance to change as well as the legacy software monopolies controlling the tooling required, as well as the cost of upskilling amongst the workforce (Charef et al., 2019; Hong et al., 2019; Barbosa et al., 2017).

While these problems can be seen and addressed individually, taken as a whole, they are symptoms of the wicked problem defining AEC’s relationship with technology. As mentioned earlier in the literature review chapter, the historical study undertaken by the BICRP in the 1970s, recommended in its final report the inclusion of sociological and organisational factors in the modernisation effort of the construction industry because of the continuous observed differences between industry-mandated formal approaches and the informal methodologies that appeared in response (Wild, 2004). Furthermore, this problem was recently resurfaced by Koch et. al. in 2019, who argue that studies in the field reveal “a piecemeal adoption of BIM and Lean” methodologies, and a general “overrating of the impact of productivity”, noting that, for example, an unresearched question is the “controlling and disciplining effect on site managers feeling forced to follow standard recipes […]” (Koch et al., 2019).

Consequently, one may claim that the state of software and its usage within AEC contrasts with today’s environment, where software is being spread “virally”, and
users become committed to the digital applications they use. Specifically aiming to counter this trend, the research principles of a living laboratory – namely, involve all stakeholders as early as possible, and experiment in real world settings – are thus an ideal methodological framework to help concurrently define and analyse the problems in the field of digital design communication in a way that does not exclude persons that are not usually involved in the research process.

3.2 Tackling a Wicked Problem

While, on an abstract level, it is safe to assume that the issue of digital design communication is a wicked problem that involves simultaneously technological, social and political issues, the work presented in this thesis is aimed at answering the three specific research questions outlined in the introduction (composable vs. complete object model, object-centred vs. file-based data classification, object-centred vs. file-based data transaction). This was achieved through a mixture of reinforcing theoretical, technical and applied analyses that were enabled by the software developed as instrumentation, named Speckle. The development of Speckle is an act of design, albeit technical, and, therefore, “wicked” and was performed within the context of a living laboratory.

It is important to note that, given a wicked problem, there is no singular “objective” resolution thereof (Rittel and Webber, 1973; Weber and Khademian, 2008). This introduces a certain bias in the analyses associated with the three research questions. To clarify, this research project does understand that technology can be seen as an application of means to achieve ends (Heidegger, 1978). Nevertheless, this view ignores the contemporary dynamic in which, to a certain extent, the digital medium has a certain agency of its own. In other words, the “ends”, or goals, are also fungible and malleable (Blitz, 2014). Therefore, when considering the software developed as research instrumentation, it is important to note that it as an active, rather than subservient or passive, actor that both serves and shapes that which it was originally designed to enable: in the present case, digital design communication.

This translates into the caveat that the problem is dependent on its solution; consequently, the research presented can be seen as biased and dependent on the way it was approached. The affordances of new digital technologies, such as internet and its communicative potential, exert a certain “push” which, when matched with the industry’s “pull”, gives rise to innovation (Pallot, 2012). Subsequently, the aim of the research questions is to identify the relevant connections between the two. This
research project attempts to abstract away potential inconsistencies; the conclusions presented should be normatively sound, and their validity should be verifiable outside the specific technical implementation of the research.

3.3 Speckle: Research Instrumentation & Industry

During the course of this research project, a digital communication software platform was developed in order to serve as the vehicle for testing theoretical assumptions in the real-life context provided by the industry partners. This research instrumentation platform, named Speckle, evolved alongside input from the InnoChain industry partner network, as well as from the open source community that grew around it.

The main goal of the Speckle platform is to enable stakeholders to send to each other design data from some of the most commonly used authoring applications in AEC. Succinctly put, it consists of a server application (the backend) that orchestrates the storage, classification and transaction of said design elements, and a host of client applications that integrate with CAD software packages and allow for the extraction of design information from within. All the interactions between the client applications and the server happen through a REST API. The client applications themselves are not restricted to CAD software connectors: they include stand-alone desktop tools developed by others, as well as web applications (SPAs/PWAs), WebGL 3D viewers, etc. (Figure 7).

![Figure 7: Schematic overview of the Speckle platform: a central REST API, implemented by a server, and consumed by a host of other client applications.](image)

The content, form and architecture of the core Speckle ecosystem evolved throughout the duration of the project—and is as such documented in Appendix 1, Tooling History. Most importantly, it underwent three complete re-writes in quick succession.
that encompassed large technical changes and shifts of focus, which reflected input from the industry, but as well due to (software) architectural dead-ends (i.e., the system would not scale to the demand of real-life projects). The analyses presented going forward do not reflect on this evolution, and rely solely on the last version of Speckle, namely 1.x.x. Currently, the codebase for the core platform components stands at approximately 54,834 lines of code. Taking into account the wider ecosystem of the platform, contributed to by the community members, it amounts to approximately 65,390 lines of code.

Throughout the project, Speckle was developed along the methodological lines of a living laboratory. Specifically:

- It was developed openly in a collaborative environment (Github), as well as it was released under a liberal open source license (MIT). This allowed the project to build the trust of stakeholders, as they would not be contributing to a project that they do not own. Subsequently, this resulted in continuous input from the technically-minded audience members. While this introduced an extra management overhead, the benefits of this approach were that the original audience expanded beyond the original InnoChain industry network, thus gaining a richer ecosystem within which the project’s research agenda could be pursued.

- Because deploying the Speckle server in the cloud represented a big technical barrier for the intended audience, it was offered as a free service solely for experimental and testing purposes. The server’s name is Hestia. By eliminating this adoption barrier, the friction in getting input from the stakeholders involved was greatly reduced. It also had the benefit of allowing non-technical persons to contribute. Lastly, it allowed for “in the wild observations” that otherwise would have been impossible.

- During late 2017 and mid 2018, as part of this project, three focus group meetings (workshops) were organised. They played a key role in validating both the technical development path of Speckle, as well as the theoretical research outcomes embodied by it. These workshops involved participants directly from the project’s research network, but were as well open to external stakeholders, forming a wide cross-section of the AEC industry (Bryman, 2012). While the face to face discussions with industry experts helped sharpen and clarify the research

---

8 As measured using the cloe command line utility, excluding metadata files and any external libraries.
9 Hestia is the ancient Greek goddess of the hearth, home and dwellings. The server can be accessed at https://hestia.speckle.works/api (accessed 12th October 2019).
and development direction, the involvement of a diverse group of end-users helped identify problems from “on the ground”; the complementary mix of roles of the focus group meeting’s participants was a key factor in revealing a full picture of the research issues addressed. The reports and minutes of these focus group meetings are presented in Appendix B, Industry Exchanges.

- Finally, during the three year duration of this project, approximately eight months were spent undergoing secondments within HENN, a large German architectural design company, and McNeel Europe, the software developers behind Rhinoceros 3D and Grasshopper, a widely used CAD application. The secondments with HENN were key in collecting requirements regarding early stage digital design communication requirements, as well as identifying key points of friction in their day-to-day design process. Furthermore, HENN served as the earliest Speckle test-bed for end users. McNeel Europe put resources forward to help with development tasks as well as end-user outreach. Furthermore, they provided invaluable input from the point of view of a developer and tool-maker, and identified key integration requirements that helped structure the shared code between software applications and Speckle.

This research instrumentation platform, Speckle, was used to test, validate and analyse alternative means of digital design communication against existing practices that are currently available to the industry. It was invaluable in collecting data necessary to answer the three research questions, both qualitative and quantitative, forming the backbone for a mixed-method approach (Bryman et al., 2008). Specifically:

- in regards with the first research question pertaining to data representation, Speckle was used to analyse the impact of how objects from existing established schemas and end-user defined ontologies are represented and used in communicative exchanges, and how they can be managed at scale; essentially, this allowed an examination of digital data representation through the interrogation of empirical data;

- in regards with the question on data classification, “live” data from Speckle\(^1\) was used to empirically compare the file-based strategy with an object-based one, assess the amount of sub-classifications a model is broken down into, as well as extract qualitative case studies supporting the research;

- Lastly, in the analysis on data transaction, it was used to quantify the impact of

\(^1\) Specifically, data was mined from the free test-server mentioned above, *Hestia*.  

69
differential (delta) updates as opposed to file-based approaches, investigate the limits thereof, as well as assess the affordances and limitations emerging from real-life usage.

By virtue of being a tangible artefact, Speckle was (and is) employed on real-life projects within the larger industry network. It thus complemented theoretical tests with the pressures of actual production use, which revealed various aspects of digital communication that informed the analyses herein. Furthermore, by developing it under an open source license, and within reach of a technical audience’s continuous scrutiny, it was able to greatly expand the size, and hence the relevancy, of the ecosystem in which the research questions have been analysed. Summing up, the methodology followed throughout this research project leveraged the academic and industrial network offered by InnoChain, and it emphasised early and intensive interactions with the stakeholders involved. In order to properly articulate a living laboratory setting, the theoretical research was complemented by a strong technical development component, which helped provide a tangible petri dish for the analyses undertaken to answer the research questions.
4. Data Representation
Previously in this thesis, the role of digital technology in communication and the effects of standardisation were discussed at a theoretical level. The AEC industries, as opposed to other design industries, are characterised by much less integrated environments in which actors are loosely coupled. The industry works on a project-by-project basis, and each project entails a different socio-technical network of contractors, sub-contractors, design firms and engineering offices. All these actors may change from one assignment to the next; even if consistency is possible, it is nevertheless not guaranteed. Design data, as such, is being produced and consumed by a diverse group of people, professions and organisations. As previously discussed, this context is fertile ground for the emergence of wicked problems. This is because even if shared values and concepts may exist, they are not guaranteed to be in place at the start of a design project. Or, for that matter, for the duration of the project itself.

In the context of digital standards for design data representation, this places an onerous burden on correctly specifying a productive level of technologically imposed precision that still allows for a heuristic approach. The correct balance between predefined formal representations and organically emerging informal standards is difficult to define in a digital context; as discussed in Chapter 2, Design and Communication section, a technical approach to communication tends to marginalise the semantic aspects of the process.

To a certain extent, this chapter asks whether one can sketch with digital representations of design data. Traditional means of design representation, such as models, sketches, plans, etc., encode design information in such a way that they deliver a certain meaning to the receiver. This meaning is not always quantifiable: for example, aesthetic qualities that can be evident in a hand drawn sketch—such as texture, material, spatial qualities—can be completely lost in a digital three-dimensional model. Such methods achieve this by speculating their medium and the context in which they are decoded and encoded to guide the receiver in inferring a certain meaning. The actors involved in the design process may not share the same internal representations of a specific building element. For example, an architect may represent a beam in one way, whereas the structural engineer in a different way. By virtue of these two persons using their own specialised software, in a digital environment, ontological difference between expert systems (domains, professions, or organisations) is a fact cemented by a technical and political reality defined by software vendors. Furthermore, the problem is compounded by the fact that, for example, an architect will evaluate a beam from an aesthetic point of view, whereas an engineer from its performance standpoint: different definitions of the same object are coupled with different notions of value.

At a purely technical level, Eastman et al. suggest that there are two major avenues
for ensuring interoperability in the AEC software world. The first one is to stay within the ecosystem of software tools provided by only one vendor, for example Autodesk or Bentley Systems (Eastman et al., 2008, p. 18). This, in theory, ensures the compatibility of all the products within that specific software “suite”. The main advantage of this approach is that it allows for “tighter and easier integration amongst products in multiple directions”, for example changes can be propagated across mechanical, architectural and engineering models. The second approach relies on standards that define building objects. The Industry Foundation Classes (IFC) object model positions itself as the official interoperability standard of the AEC industry, and has been in active development since 1994 (Froese, 2003; Plume and Mitchell, 2007; van Berlo et al., 2012). It is widely used throughout the whole design and build process and underpins most, if not all, BIM techniques, regulations and practices. IFC classes aim to specify a complete vocabulary of all elements (both high level—walls, doors, etc.—as well as low level—points, lines, etc.) that are used in architectural design, engineering and construction (Hamil, 1994). While the “universality” of IFC as a standard is recognised throughout the industry, its actual implementation within the competing CAD software packages is anything but. Software vendors rely on custom specialised features to differentiate themselves, thus framing a conflict of interest between the needs of their users and their own business survival. Consequently, this has resulted in incomplete and inconsistent implementations of the IFC specifications throughout the CAD software industry that negate the benefits that come from standardisation: because of differences in how IFC data is produced and consumed, its reliability is compromised.

Because of these frustrations, and coupled with the fact that the design process becomes more and more digitalised and data-oriented, many engineering and design companies have been developing in-house alternative design data exchange standards positioned as internal replacements for the official IFC standard, rather than as additions or extensions. These replacements aim to better suit their own need and have applicability scopes that vary from a single-project base to a company-wide norm. The emergence of such standards results from the need to further articulate domain- and organisation-specific knowledge that accumulates over time: most observed examples underpin computational techniques, workflows and methodologies that quintessentially represent a given organisation market advantage, or a given domain’s internal language constructs that allow for its effective functioning (Giddens, 1991; Kuhn, 1962). As reflected in Chapter 2, Literature Review, communication is an inferential process during which the cognitive processing of the communicants tends to be minimised—which is a fundamentally dynamic process in direct conflict with the rigidity of a standard.
Consequently, one can argue that the emergence of other object models describing specialised subsets of the objects that compose the built environment is a recurring natural phenomenon of ontological revision.

In the field of computer science, the development and increasing adoption of self-describing structures for object representation, such as JSON, enable alternative and more expressive ad-hoc “object models” to be defined. These allow for the representation of information to evolve throughout time easily in a coupled way with its implementations in various consuming applications. At the same time, they allow for the crystallisation in given patterns once a consensus, or balance, is reached. This chapter investigates if a similar approach to data representation is valid within the AEC industry: abstract enough to allow for the encoding and decoding of multiple ontologies, flexible enough to facilitate their revision and their combination, and thus supporting the emergence of shared meaning and understanding at a representational level.

4.1 Composable Data Structures

Throughout the course of this research project two different approaches have been experimentally enabled and assessed. The first approach, namely that of composable data structures, started on the assumption that all the potential elements of the built environment can be ultimately reduced to and defined by a standard set of geometric objects—points, lines, circles, polylines, etc.—and the usual programming language primitives—booleans, numbers, strings. Furthermore, this approach does not carry the assumption that one is able to predict in which way end users compose them (or relate to each other, or structure them internally) into higher level design objects such as walls, beams, columns, doors, etc: each of these higher-order ontological definitions, while similar at a superficial glance, can hold different meanings and internal representation structure depending on the domain that they are interpreted. For example, as mentioned above, a beam is a volumetric object for the architect, but a one dimensional line for the structural engineer. Similarly, one cannot know what properties a wall element will be best described by: it can be represented by a centre line, height and width for the purpose of spatial design; or it might be represented as a purely volumetric element for the purpose of costing; or it might be represented as an assembly date, a delivery date and site storage information if it is a prefabricated element. In a multi-disciplinary context, these object definitions can have partially shared properties, but their overall consistency is not guaranteed as the project evolves.
4.1.1 Implementation

In order to test this approach, a set of basic geometric object primitives were specified using the OpenAPI language. Subsequently, these were implemented in the .NET (C#) framework in order to be embedded in existing CAD applications environments. Specifications are formal descriptions of the requirements that an implementation needs to meet. Subsequently, the implementation itself can be enacted in any number of programming frameworks as long as it meets the requirements of the specification it is based on. This is relevant because one cannot assume a common programming environment shared by the digital software tools that produce and consume design data. While an assumption as to the fact that they may share, to a certain extent, basic definitions of geometric primitives (e.g., a point will be defined by three coordinates, a line by two points, etc.), the various software used in the industry are built on top of different code platforms (e.g., Rhinoceros 3D is based on the .NET framework, whereas SketchUp can be extended via Ruby).

The deciding factor in choosing the OpenAPI framework as a base in which to write the specification, against EXPRESS, RAML and API Blueprint or others, was based on several facts. First, it can be used to specify both server protocol implementations (which allowed us to have one specification document for the entirety of the Speckle platform) as well as client implementations. Second, compared to the other alternatives, OpenAPI specifications benefit from a wider range of tooling that can generatively scaffold said implementations, thus partly automating the process of writing code and simultaneously ensuring consistency across said programming environments. Finally, the community and overall project health of the OpenAPI framework is generally perceived as lending it good outlook in terms of future continuity and support.

Consequently, Speckle, implementing the composable data structures approach, allows for object definitions that can be arbitrarily extended and composed by the end users. “ Arbitrarily” signifies the ability to continuously inform and adapt the definition of a given design element to the various ontologies of the actors interacting with said element. This is achieved, at a higher level, by allowing users to add semantic triples (Cyganiak et al., 2014) to the basic set of predefined object primitives mentioned above. A semantic triple codifies the relationship between two entities in the form of “subject – predicate – object” expressions, thus expressing informational relationships in a machine and human readable way. For example, one can codify the definition of a structural beam object by specifying its end point, start point, structural type, restraints and loads, or as well by its axis line, orientation and section property if the latter definition is better suited than the former to the task at hand.
Using dot notation to express the above, one can define such a triple as:
myBeam.startPoint = myStartPoint. In this example, myBeam is the subject, startPoint is the
predicate and myStartPoint is the object. Dynamically typed languages, such as
JavaScript or Python, allow for the dynamic creation of such triples on any object
without them being defined on its base class. Nevertheless, statically typed
languages, such as C#, do not easily allow for this behaviour. Consequently, in the
specification and subsequent implementation of the base geometry and primitive
classes, the technical implementation opted for a compromise whereby custom
properties can be defined inside a designated field named properties. This specific
field can then be implemented as a specific dynamic structure native to coding
framework itself (e.g., in JavaScript, it will be simply another default object, as all
objects are key-value pairs, whereas in C#, it will be implemented as a
dictionary<string, object>, which is one of the key-value pair primitives offered by
the language.

The list of primitives that have been defined throughout the course of this project, at
the time of writing, comprises the following 20 types: Boolean, Number, String,
Interval, Interval2d, Point, Vector, Plane, Line, Circle, Arc, Ellipse, Polyline,
Curve, Polycurve, Box, Mesh, Extrusion, Brep, Abstract

This set of object types can be split into two categories, namely (1) primitives in the
classical understanding of computer science – specifically, boolean values (true or
false), numbers (defaulting, in practice, to the programming language’s
implementation, otherwise a double-precision floating-point format, occupying 64
bits of computer memory) and strings (UTF-8 encoded); (2) basic mathematical
constructs used extensively in computational geometry, specifically one- and two-
dimensional intervals, vectors and planes; and (3) actual computational geometry
primitives, namely lines, circles, meshes, etc. While the internal definitions of these
objects have evolved in time driven by user needs, the actual set was chosen based
on the fact that most used authoring software supports them either as first-class
objects, or as supporting types underpinning higher level objects. For example,
Grasshopper and Dynamo have the concept of one- and two-dimensional intervals,
which is used in surface subdivision computations; Revit, while having an XYZ
construct representing a Point, does not natively support them as document objects –
they are used exclusively in scaffolding the basic shape and form of higher level

11 The full implementation is available online at the following link:
https://github.com/speckleworks/SpeckleCoreGeometry/blob/480a52b69a5f8b7c98c4610cd69d612b
7353fe2b/SpeckleCoreGeometryClasses/Base.cs (accessed 2nd June 2019).
objects, such as Walls, AdaptiveFamily, etc.

```
"SpeckleMesh": {
  "allOf": [
    {
      "$ref": "/definitions/SpeckleObject"
    },
    {
      "type": "object",
      "properties": {
        "type": {
          "default": "Mesh"
        },
        "vertices": {
          "type": "array",
          "items": {
            "type": "number"
          }
        },
        "faces": {
          "type": "array",
          "items": {
            "type": "number"
          }
        },
        "textureCoordinates": {
          "type": "array",
          "items": {
            "type": "number"
          }
        }
      }
    }
  ]
}
```

**Figure 8:** Above: the specification entry for the Mesh geometric primitive. Below: the implementation in C# of the Mesh class, automatically generated from the specification above.

```csharp
public partial class SpeckleMesh : SpeckleObject
{
  public override string Type { get => "Mesh"; set => base.Type = value; }
  public List<double> Vertices { get; set; }
  public List<int> Faces { get; set; }
  public List<double> TextureCoordinates { get; set; }
}
```
Each of these objects have their own internal properties that describe its structure in a minimal way, and they share a set of common fields. The most important of these are `objectId` and `hash`; two equivalent object identity mechanisms, both used in the persistence and classification layer discussed in *Chapter 5, Data Classification*, as well as for purposes of data deduplication and diffing which are discussed in *Chapter 6, Data Transaction*. The former, `objectId`, is generated exclusively by the persistence layer (the logic and hardware pertaining to data storage from a software project), whereas the latter, `hash` can be generated both at the application layer (the part of a software project concerned with “business logic”) as well as at the persistence layer, only if it is not present. This is because, while the hashing algorithms used are the same (MD5), the availability of data from which to compute an accurate hash is lower at the persistence layer.

The `hash` property, essentially, represents a unique string that is specific to a given object’s state. For example, a point object defined by `[ 0, 0, 0 ]` (`hash = 13e81f6567b656a19c629377c7f5a698`) will have a different hash if its coordinates are changed to, for example `[ 1, 0, 1 ]` (`hash = 53ed8181875a36311d34ba1b5b46ff29`). The hash is generated by passing the object’s byte array footprint from the computer’s memory to a standard hashing function, namely MD5. The resulting byte array is then converted to a hexadecimal string. The role of a hashing function is to map arbitrarily sized data to fixed size values. MD5, while proven to be cryptographically insecure, is a suitable choice given its efficient implementation on numerous platforms, low collision rate\(^\text{12}\), and given that it is not used in this context for cryptographic purposes.

These class definitions were grouped under the name of `SpeckleCoreGeometry`. Several other complementary assemblies were developed to facilitate the encoding to and from the various CAD applications, namely Rhino, Grasshopper, Dynamo and Revit. Specifically, they facilitated the low level translation between the data representation of, for example, a Rhino Mesh object and a Speckle Mesh object, or a Dynamo Point to a Speckle Point. These assemblies dynamically extend the original object model with conversions methods that were as well dynamically loaded at runtime, thus allowing for injecting at runtime the correct dependencies in the host application itself. As such, when the Speckle plugin would be loaded by, for example, Dynamo, an end user would be able to access a `ToSpeckle()` method on all Dynamo geometry primitives.

\(^{12}\) Collisions may occur after storing an excess of \(2^{64}\) hashes, or 18,446,744,073,709,600,000.
that are supported, and a `ToNative()` method for all Speckle primitives that can be translated, or cast to native Dynamo objects.

### 4.1.2 Applications

Throughout the duration of this research project, the affordances offered by the composable object model approach described so far were used often to customise and enhance the digital data exchanges between the various sub problems of a given design task. For example, in a multidisciplinary team working on a façade project, the architectural stakeholders would use rectangular polyline-based definitions for the glass panels. The structural engineers would subsequently expand the object definition of a façade panel to contain various analysis results, such as end reaction forces on the four structural nodes. Nevertheless, these were used internally in their design process to size structural elements. When the resulting data was being fed back to the architects, the definition of a façade panel mutated again: the structural properties were partly replaced by a different set of properties, this time reflecting the volumetric properties of the newly designed structural support members as well as the tolerances and offsets needed for each particular panel.

Furthermore, composability opened the possibility of merging together multiple object-models. In other words, it allowed end-users to combine one object model with another one. For example, consider the representational definition of a structural beam, one useful for analysis purposes—centre line, loads, degrees of freedom of connection points—and a separate, product representation that would be fit for construction and logistical evaluations. Both these definitions could come from different object models, each defined by a specific stakeholder group, nevertheless they could coexist within the same object. These composite objects have the possibility to share basic traits, if a common set of properties is present.

Paul Poinet, a fellow InnoChain researcher, has advanced in this direction by creating a user interface that allows for the arbitrary construction of relationships between objects directly in Rhino, as well as adding custom, user defined properties (Poinet et al., 2018). Furthermore, his experiments with multiple-schema objects—involving a beam defined simultaneously from an engineer’s point of view, using an object model developed by a structural engineering company, and from a fabricator’s point of view (Figure 9)—demonstrated the potential of this approach to bridge information and meaning across domains of expertise.
By allowing users to interactively compose higher level objects based on a low-level schema, this approach distances itself from the rigours of a standard, and, most importantly, encourages dialogue to emerge between the actors involved in the communication process. The diagram below shows such a scenario (Figure 10): an actor (Mary), in Context A, creates a specific type of object, namely CustomObject_A and transmits it to a different actor (Alex) in Context B. Nevertheless, Alex does not share the same definition of that specific object as Mary does; or either he might need, for
his purposes, to add, remove or combine properties of \textit{CustomObject\_A} in various ways to suit his own speciality, thus creating a new derived object type, namely \textit{CustomObject\_B}. This process can be repeated, with Alex further revising his object definition in a different context, for instance Context \textit{C}, where different key-value pairs can be added or removed dynamically to better match the internal representation of the object with the ontology of the specific domain, or task, within which it is embedded. An example of such a customised object is shown in Figure 11, where the default properties are highlighted against the ones added by an end-user.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure10.png}
\caption{Ontological revision process: objects can be packed, unpacked and re-defined in various contexts by their users.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure11.png}
\caption{Example diagrammatic representation of user defined objects.}
\end{figure}
4.1.3 Affordances and Limitations

One of the major limitations of the composable data structures approach is the fact that there is no evident way to instrument this behaviour globally, across the various CAD software used by the AEC industry. This is because there is a tight coupling between the affordances of the user interface, software environment, and the actual process needed for composing new, higher-level objects. Within a WYSIWYG environment, such as direct modelling in Revit, or Rhino, exposing such a functionality would require a large investment in the development of the user interface that would enable end users to employ it. Nevertheless, visual programming environments, such as Grasshopper and Dynamo, allowed us to instrument this process of object definition based on ontological revision by developing custom components that facilitated the process of adding, removing, and nesting user-defined key value pairs.

In this specific context, a composable object model proved to be an extremely productive vehicle for ontological revision. Specifically, by allowing end-users to freely exchange, combine and create temporary new ontologies as dictated by their need, the domain specific processes were better and more efficiently served. Essentially, a composable object model allowed for linkages between two, or more, separate domains to mutate over time, and thus build up and evolve a shared set of digital representations of design data.

4.2 Encoding Existing Ontologies

This section tackles the issue of existing object models and can be referred to as an attempt to explore whether the flexibility of the composable approach described above can be programmatically retro-fitted to existing well-defined schemas. As opposed to defining object ontologies “on the fly”, whereby end users create and evolve a glossary of building elements or concepts that is informal, or in other words not formally recorded in a central place, many AEC companies and organisations have created object models that encode pre-defined ontologies in a formal, centralised manner.

The most known is the IFC standard, the development of which is the purview of BuildingSmart International, a consortium of software development companies as well as industry representatives. During the course of this research, other object models developed internally within the various industry partners that this project
attracted were observed\(^{13}\). Amongst these there are BHoM (The Buildings and Habitats Core object Model, 2019), Pegasus and DesignLink (Holzer and Downing, 2010). These object models, much smaller in size and ambition, were developed primarily to resolve the issues of interoperability that the IFC standard did not solve due to its fragmented implementation. As such, they were limited in their overall vocabulary, but efficient in solving the problems for which they were designed and streamlining the data exchanges that underpinned their authors’ design processes.

The original experiment with composable object models was well received by the project’s industry partners. Nevertheless, given their reliance on previously developed object models that already structured data in a way that facilitated their design and analysis processes, it was important to investigate whether existing schemas could be programatically supported by dynamically translating them to use the same triple mechanism that was employed by the composable approach. Consequently, this section looks at the affordances and limitations of encoding and decoding an existing well defined object model to and from the composable object model described in the previous ly.

4.2.1 Implementation

As all of the object models that were accessible from the living laboratory context were based on the .NET framework and ultimately compiled to assemblies distributed via dynamically linked libraries (.dll), the following technical implementation is similarly applicable only in the mentioned programmatic context. At its core, the implementation relies on the ability to reflectively inspect the properties (or fields) of an object of any type without being aware of its class definition (its specification), which is present in its original, potentially locked down, assembly. The basic object schema definitions, described in the previous section, were extended so as to contain a new type of object (SpeckleAbstract) which would specifically be able to host the information from any existing object. First, its properties and their values, broken down into the primitives that underpin the programming language, will be stored in the properties field, similar with the case for custom user-defined fields on base objects. Second, the name of the original schema from which the object came from, will be stored in a special, hidden field (_type) in a .NET framework specific format.

The process of creating a dynamic object out of a predefined one is as follows: for a given object, retrieve its existing public properties. For each of these properties, check

---

\(^{13}\) At a cursory glance, data gathered shows more than 20 different unique object models.
whether the value of the property is a primitive object (boolean, number, string, or any of the geometric primitives defined). If this condition is satisfied, then create a new triple (key-value property) in the SpeckleAbstract’s properties dictionary with the key being equal to the original’s object property name, and the value being the one extracted from its instance. If the value does not meet the primitive object criterion, and thus is a different sub-object, create a new key-value property, with the key being equal to the original’s object property name, and set its value by repeating the process described, but applied on the sub-object.

```csharp
public SpeckleAbstract EncodeAbstract( object myCustomObject )
{
    var mySpeckleAbstract = new mySpeckleAbstract();
    mySpeckleAbstract._type = myCustomObject.GetType().GetAssembly().Name;

    foreach( var property in myCustomObject.GetType().GetProperties() )
    {
        var myValue = property.GetValue();
        if( myValue is Primitive )
            mySpeckleAbstract.properties[ property.Name ] = myValue;
        else
            mySpeckleAbstract.properties[ property.Name ] = EncodeAbstract( myValue );
    }
    return mySpeckleAbstract;
}
```

*Figure 12: (Code block) Pseudo-code implementation of the encoding method for a SpeckleAbstract object.*

Figure 12 shows a pseudocode rendition of the encoding method. The actual implementation\(^{14}\) is much more complex, as it needs to handle various specific cases – specifically, instances of properties that are lists (IEnumerable) or dictionaries (IDictionary), but as well potential recursive references. Furthermore, because the creation of this object is done programmatically, and not by invoking a constructor, its hash is also set by generating it from the object’s binary representation, extracted from its memory footprint.

\(^{14}\)The full code is available here: https://github.com/speckleworks/SpeckleCore/blob/9545e96f04d85f46203a99c21c76eaa0e03dae/SpeckleCore/Conversion/ConverterSerialisation.cs#L39-L198 (accessed 2\(^{nd}\) June 2019).
4.2.2 BHoM & Pegasus

Buro Happold, an InnoChain industry partner, has been developing its own internal set of computational tools to standardise and speed up internal processes of design and analysis. These tools are grouped under the name of BHoM. It consists of an object model containing definitions for various classes that describe categories of objects grouped under various categories depending on their use case (Geometrical, Acoustic, Structural, Architectural, Planning, etc.—there are a total of approximately 273 classes and interfaces). These comprise an evolving ontology that allows for the standardisation of the company’s digital processes across multiple software packages. In short, BHoM can be seen as an extensible object model which underpins the internal processes of interoperability and specialised computation. It allows users to employ various specialist software in a platform-independent way by assembling digital analysis workflows in a set of pre-existing parametric modelling user interfaces, such as Grasshopper and Dynamo.

Dialog, an engineering and design company with offices around North America, have been integrating Speckle at the core of their digital processes. Similarly to Buro Happold, they have also been developing a set of internal software tools that automates and unifies their analysis processes. These set of tools revolve around a core object model, namely Pegasus, and various integrations with other CAD/CAM software packages, such as ETABS, Robot, Rhino, Unity, etc.

During the course of this research project, they relied on Speckle’s compatibility with existing well defined schemas to be able to transfer data between the application clients that they have developed to match their internal processes. Dialog pursued an independent, parallel effort to that of Buro Happold because, at the time, BHoM was not yet released to the public, or open sourced. Nevertheless, the relevant differences between the two object models are minor – they were strikingly similar as they were created to cater to similar engineering needs. In the descriptions that follow, BHoM is the primary reference implementation, but the same analysis and reflection can be applied to Pegasus.

The architecture of BHoM’s codebase is well defined: object definitions are logically separated from operations that execute actual computational work on them. Furthermore, BHoM object definitions follow a simple and clear inheritance and composability pattern, and have an ideal depth of inheritance and class coupling index – both fitting within what is accepted as “good practice”. This allowed us to assess the viability of Speckle’s serialisation (encoding) and deserialisation (decoding) routines in handling Buro Happold’s object model.
As an experimental setup, a Bar object, from the BHoM’s structural elements object model subsection, is programmatically instantiated. This element is composed of two nodes (start and end points) and several other classes and enumerations. It does not contain any circular references, and neither have any direct translations routines to SpeckleObjects been added—there is no special conversion routine employed. Its diagrammatic representation as a SpeckleAbstract object is shown below (Figure 13), and its full composition revealed.

![Diagram of BHoM Bar object]

Figure 13: A graph showing the intermediate representation of a BHoM Bar object, as it was cast into a SpeckleAbstract DTO.

Within a context where there is access to the BHoM package, the deserialisation routine performs as expected: it is able to convert the resulting composite SpeckleAbstract object into its native BHoM object type (Bar) and its subcomponents (Node and Point), without any loss of fidelity. The same is achieved for all other tested object types; the verification methodology is to serialise the original object, deserialise it and then re-serialise to a SpeckleObject and check if the generated unique object hashes match.

The BHoM schema is user extendable and there are scenarios where circular or linear references are possible. For example, for the sake of convenience, the following scenario is envisioned: the GSA analysis software defines structural elements by their direct spatial coordinates, and therefore the author of the information adds direct references to them in order to have easier access without nesting. Consequently, the handling of linear references is tested by adding a set of custom properties to the Bar that reference some of its internal parts, namely EndNode, StartNode, and their
respective \textit{Location} properties. A diagrammatic representation of the resulting \texttt{SpeckleAbstract} object is below (Figure 14).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure14.png}
\caption{A diagrammatic representation of the reference tree of a given BHoM Bar object. The black circles represent object properties, the large grey circles represent embedded objects, and the arrows indicate nested references.}
\end{figure}

As the diagram above shows, the serialisation routine correctly identifies them and instead of copying the objects, sets a reference to the originals. Upon deserialisation, these are linked back to the newly created internal objects.

A recursive reference can happen if a sub-object contains a link to its parent object. For example, consider a set of \texttt{Nodes} that each have reference to their parent \texttt{Bars}. A corner node can thus reference two different \texttt{Bars}. Similarly, the \texttt{Bar} object holds two references to two different \texttt{Nodes}, which, in turn, reference their \texttt{Bar}. This can lead to cases in which, when serialising an object, if it has a large dependency tree, the routine may end up failing due to a too high recursion depth. The diagram on the left in Figure 15 illustrates such a potentially dangerous scenario, whereas the one on the right illustrates a “benign” case.
Figure 15: Diagram showing a generalised case of recursive references with a given object. Left: deeply-nested references; Right: simplified example, showing the child nodes directly referencing their parent.

In order to prevent endless recursion, the encoding routine for SpeckleAbstract objects stops at a stack depth of eight\textsuperscript{15}. This results in the fact that objects that are too deeply nested into each other will not get serialised. This places an onus on the end users and, most importantly, developers to ensure that a more atomic structural model is observed\textsuperscript{16}.

### 4.2.3 GeometryGym IFC (ggIFC)

GeometryGym provides a vendor independent implementation of the IFC object model that works with a number of software packages, namely Rhino, Grasshopper, Revit, Tekla, etc. It is one of the most widely used interoperability solutions that is not bound to one software ecosystem or another. With the help of Jon Mirtschin, the author of the ggIFC toolkit, this section investigated the potential for IFC objects to be successfully encoded and decoded using the “schema-agnostic” strategy.

Initial tests show that, because of the way the IFC schema is specified and subsequently implemented, serialising one object will result in capturing an extremely long dependency chain of inter-related objects. If one suppresses the maximum serialisation recursion depth, it will essentially amount to the whole “file”. This is because any IFC object contains a chain of references that point to an IFC “root” element, which thereafter links to all other IFC objects present. In contrast with the BHoM object model, where this behaviour may happen but is by no means enforced or encouraged, the IFC schema mandates this tight coupling of objects to

---

\textsuperscript{15} This number is, in programming lingo, a \textit{magic number}. In other words, it has been chosen heuristically according to observed behaviour (allowing for a reasonable amount of depth, but protecting from too nested objects), but it is not supported by evidence and does not imply code safety, correctness or utility.

\textsuperscript{16} In a later implementation, this limitation is removed by relying on an array that kept track of previously processed objects. Simply put, if a previously processed object is encountered, the respective branch is no longer followed.
their surrounding context.

An example diagrammatic view of a resulting SpeckleAbstract object representing a ggIfcGrid in a blank file is shown in the right-most diagram below (Figure 16); the sheer size of the object tree is a direct result of the schema’s tight coupling. Because this was not a practical outcome, Jon Mirtschin, by changing GeometryGym’s implementation of the IFC standard to exclude object properties that create links to the rest of the whole from the serialisation routines, shows that it is possible to properly serialise a singular IFC object using the default Speckle encoding routines. This is tested and validated by serialising an ggIfcGrid created in Grasshopper and subsequently deserialising it as a native Autodesk Revit GridLine element.

![Diagram of object trees](image)

**Figure 16**: Evolution of the ggIFCGrid object representation: the right hand side version is the original, whereas the left hand side version is the optimised version.

Further refinement, authored by Jon Mirtschin, on the way IFC objects are specified and subsequently serialised led to an even smaller and more efficient representational value of the ggIfcGrid (see above in Figure 16). It is important to note that this strategy relies on Speckle’s serialisation and deserialisation routines that produce SpeckleAbstract objects, nevertheless there is custom application specific code needed for embedding the IFC objects back into Revit that is handled by the GeometryGym toolkit, and masks extra complexity.
4.2.4 Affordances and Limitations

The approach proposed in this section, namely that of programatically matching strict, well-defined object models with the previously created composable object schema, is highly dependent on how the target object model has been authored. Specifically, the two most important metrics that affect the behaviour and size of a SpeckleAbstract object are a) the depth of inheritance and b) the class coupling.

Depth of inheritance (DI) is a measure that captures the “maximum length from the node to the root of a tree” (Chidamber and Kemerer, 1994). In other words, it measures how much classes inherit from each other. For example, if one has a class called Car that inherits from a more generic one, named LandVehicle, which in its turn inherits from the even more generic TransportationMean, the object model will have a DI factor of three. Chidamber and Kemerer posit that the higher the DI factor of a class, then

- the more methods it is likely to inherit, thus making it less predictable,
- the greater the design complexity (negative), and
- the greater the potential reuse of inherited methods (positive).

The second metric, class coupling (CC), assesses how interwoven—or interdependent—a given object model is. It does so by averaging, throughout the codebase, how many secondary classes one class is dependent upon. As such, for example, if the Bar class contains properties defined from three different other classes, it will have a score of three. While the previous measure is ambivalent, class coupling is seen as a negative property—the higher the measure, the lower the quality of the code (Chidamber and Kemerer, 1994) due to the potential for error. The table below presents the analysis results from Microsoft’s CodeMetrics software on the BHoM (The Buildings and Habitats Object Model, 2019), GeometryGym IFC (GeometryGym OpenBIM IFC, 2019) and IFCKit, BuildingSmart’s official C# implementation and specification of the IFC standard (IFC Documentation and Toolkit, 2019) (Figure 17).

<table>
<thead>
<tr>
<th></th>
<th>Depth of inheritance (DI)</th>
<th>Class Coupling (CC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BHoM</td>
<td>3</td>
<td>51</td>
</tr>
<tr>
<td>ggIFC</td>
<td>11</td>
<td>1132</td>
</tr>
<tr>
<td>IFCKit</td>
<td>9</td>
<td>693</td>
</tr>
</tbody>
</table>

Figure 17: Comparative table showing the depth of inheritance and class coupling metrics for three existing object models.
From the table above, one can see how the IFCKit object model has a lower overall complexity as compared to ggIFC. Nevertheless, ggIFC is much easier to use than the standard IFCKit implementation: the difference in depth of inheritance can be attributed to additional user-facing functionalities, such as accessible object constructors and extra serialisation routines that support multiple protocols. The BHoM object model, being smaller in size and scope, scores much lower than the other two and consequently lends itself much easier to the approach presented in this section. The two IFC-derived object models, due to their DI, and especially, their CC score, generate large, unwieldy and needlessly complex objects that, while still machine readable, not particularly fit for human apprehension.

Class coupling can be tied to one of the biggest critiques of object-oriented programming. Joe Armstrong, creator of the programming language Erlang, famously stated that “[…] the problem with object oriented languages is they’ve got all this implicit environment they carry around with them. You wanted a banana, but what you got was a gorilla holding the banana and the entire jungle.” (Seibel, 2009)

This issue was clearly visible in the early tests of encoding ggIFC objects, and the refactoring effort that its author undertook was directed at minimising and managing the class coupling of the object model so that its encoding and decoding to and from Speckle became feasible.

This observation is confirmed by an empirical analysis of the data aggregated in the experimental Speckle server, Hestia, which was made freely available as part of the living laboratory context. Out of 152,200 SpeckleAbstract objects in total, derived from a total of 24 unique object models, the table below compares the average tree depth17 and size (expressed in bytes)18 of the existing 2,026 ggIFC-derived and 5,769 BHoM-derived objects (Figure 18).

<table>
<thead>
<tr>
<th></th>
<th>Count</th>
<th>Average Tree Depth</th>
<th>Average Size (bytes)</th>
<th>DI</th>
<th>CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>BHoM</td>
<td>5,769</td>
<td>6.03</td>
<td>2,935.00 (~3kb)</td>
<td>3</td>
<td>51</td>
</tr>
<tr>
<td>ggIFC</td>
<td>2,026</td>
<td>16.93</td>
<td>36,161.99 (~36kb)</td>
<td>11</td>
<td>1132</td>
</tr>
</tbody>
</table>

Figure 18: Comparative analysis between the average tree depth and size of BHoM- and ggIFC-derived SpeckleAbstract objects.

---

17 The tree depth of an object is obtained by calculating the maximum nesting levels of its properties.
18 The object’s size was calculated by measuring the byte array length resulting from its JSON representation.
The table above reveals that BHoM-derived objects have a much lower average tree depth and size compared to ggIFC-derived objects. The average tree depth of IFC based objects is approximately three times larger than that of BHoM based ones (16.93 vs. 6.03). This has an important cognitive implication on the process of ontological revision: as mentioned previously, the shallower a data structure is, the easier it is for an end-user (developer or, in other cases, designer) to iterate and improve upon it. Lastly, there is a striking difference in the average size of objects: ggIFC-derived instances are, on average, 12 times larger than BHoM-derived ones. This difference in size, when extrapolated to more extensive usage scenarios, leads to potential huge savings in overall storage size.

While the tests based on the IFC object model yielded mixed results, BHoM, and Pegasus are examples where this second approach demonstrated the viability of programmatically matching the flexibility of generating “on-the-fly” schemas with the properties of a formally predefined schema. The main difference being that, instead of having the user manually create custom object properties, these would be automatically instantiated from a given schema definition. Summing up, this proves that a composable object model can, programmatically, lend itself to both end-user “improvisation”, which facilitates a faster rate of ontological revision, as well as a pre-defined fixed schema which satisfies the different need of consistency throughout the design process at later stages.

4.3 Comparative Analysis of Data Structure Sizes

Within the database of the Speckle server Hestia, there are, at the time of writing, approximately 16,834,814 objects\(^{19}\). Based on this data set, an empirical analysis has been conducted that allows us to assess the structure and size of the objects resulting from real-life usage of Speckle. The table below shows the average tree depth and object size of three distinct object categories: (1) abstract objects, derived from existing ontologies, (2) objects with user defined ontologies, and (3) all other objects (not abstract, and with no user defined properties). For reference, the statistics from the previous figure regarding BHoM- and ggIFC-derived abstract objects are replicated in the last two rows (Figure 19).

---

\(^{19}\) At the time of review, circa two months later (October 2019), there currently are 20,795,069 objects. At the time of submission (August 2020) this number exceeds 40 million.
<table>
<thead>
<tr>
<th></th>
<th>Count</th>
<th>% of total</th>
<th>Average Tree Depth</th>
<th>Average Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract Objects</td>
<td>152,220</td>
<td>0.9%</td>
<td>3.43</td>
<td>7,738.27 (~7.7kb)</td>
</tr>
<tr>
<td>User Objects</td>
<td>3,099,435</td>
<td>18.41%</td>
<td>2.97 (1.97)</td>
<td>1,095.45 (~1.0kb)</td>
</tr>
<tr>
<td>All others</td>
<td>12,340,793</td>
<td>80.68%</td>
<td>1.42</td>
<td>4,523.29 (~4.5kb)</td>
</tr>
<tr>
<td>BHoM (Abstract)</td>
<td>5,769</td>
<td>&lt; 0.01%</td>
<td>6.03</td>
<td>2,935.00 (~3kb)</td>
</tr>
<tr>
<td>ggIFC (Abstract)</td>
<td>2,026</td>
<td>&lt; 0.01%</td>
<td>16.93</td>
<td>36,161.99 (~36kb)</td>
</tr>
</tbody>
</table>

*Figure 19: Comparative analysis table of object depths and sizes.*

The most striking finding is the overall high percentage of objects with user-defined properties (18.41%). This suggests a strong user inclination towards engaging with custom data structures and reinforces the hypothesis that revision on an ontological level is crucial. This observation can be partly biased: the creation of such objects is currently facilitated by only two Speckle plugins, catering for Grasshopper and Dynamo, which were, for a long time, the only ones available. These two visual programming software are primarily used by knowledgeable “power users” who are prone to customising available toolsets. Nevertheless, it proves that ontological revision is a key aspect of design communication: almost one in five objects have been customised by end users.

User defined objects (as described in *Section 4.1*) have an average tree depth value of 2.97. This value takes into account the host object, as well as the properties filed that the user customised. Consequently, the average depth of the exclusively defined by an end user is 1.97 (subtracting the one hierarchical level given by the base object). This informs us with regards to the nature of the process of ontological revision from purely the end-users’ point of view: complex, deeply nested structures (with higher average tree depths) are shunned in preference for shallower, more flat hierarchies of data (with lower overall tree depths).

Coupled with this finding is a distinct reduction in the average size of objects: those derived from pre-existing ontologies tend to be 7 times larger than those defined in an ad-hoc manner by end-users (7.7kb for *Abstract* objects vs. 1.0kb for user-defined objects). The overall average size of an ontologically derived object is 4.5kb, four times larger than the average user defined one. Consequently, this results in a leaner communication volume, and important savings in storage. These findings have an
important bearing on the way data is classified and transacted (discussed in *Chapter 5, Data Classification*, and *Chapter 6, Data Transaction*, respectively).

The table below provides an overview of the type of base objects that were customised by end users (Figure 20). It shows that, for example, only 172 Box type objects have been enriched with custom information, as opposed to 847,632 Mesh type objects and 1,310,233 Point type objects. Following this, this analysis continues by further comparing the number of customised objects against the overall count of that object’s specific type from Hestia’s database.

<table>
<thead>
<tr>
<th></th>
<th>Enriched Count</th>
<th>Overall Count</th>
<th>% Enriched</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plane</td>
<td>25</td>
<td>71,949</td>
<td>0.03%</td>
</tr>
<tr>
<td>Box</td>
<td>172</td>
<td>177,594</td>
<td>0.10%</td>
</tr>
<tr>
<td>Circle</td>
<td>647</td>
<td>43,141</td>
<td>1.50%</td>
</tr>
<tr>
<td>Polycurve</td>
<td>2,474</td>
<td>67,831</td>
<td>3.65%</td>
</tr>
<tr>
<td>Object</td>
<td>15,612</td>
<td>15,612</td>
<td>100.00%</td>
</tr>
<tr>
<td>Ellipse</td>
<td>44,136</td>
<td>49,498</td>
<td>89.17%</td>
</tr>
<tr>
<td>Arc</td>
<td>96,740</td>
<td>254,591</td>
<td>38.00%</td>
</tr>
<tr>
<td>Curve</td>
<td>125,795</td>
<td>173,810</td>
<td>72.38%</td>
</tr>
<tr>
<td>Brep</td>
<td>215,015</td>
<td>818,745</td>
<td>26.26%</td>
</tr>
<tr>
<td>Polyline</td>
<td>217,082</td>
<td>793,989</td>
<td>27.34%</td>
</tr>
<tr>
<td>Line</td>
<td>223,872</td>
<td>4,968,801</td>
<td>4.51%</td>
</tr>
<tr>
<td>Mesh</td>
<td>847,632</td>
<td>1,517,236</td>
<td>55.87%</td>
</tr>
<tr>
<td>Point</td>
<td>1,310,233</td>
<td>6,064,101</td>
<td>21.61%</td>
</tr>
</tbody>
</table>

*Figure 20: Table representing the number of object types that have been customised by end users.*

By actual volume, Points are the most customised object type present, followed by Meshes and Lines. Nevertheless, by percentage out of the overall count of an object’s specific type, Ellipses, Curves, Meshes, Arcs and Polylines all surpass Points: these geometrical “atoms” were used in a more exclusive manner, specifically in specialised workflows. From live observations of the usage of Speckle in practice, ellipses, nurbs curves and arcs are used in bypassing a major interoperability hurdle, namely that amongst free form volumetric NURBS objects and, furthermore, they were often the base upon which complex geometry was rationalised for manufacturing or other later design stages.

The outlier by percentage, the Object type, is explained by the fact that it exists purely
as blank slate for end users to create their custom ontologies; therefore, all of its instances are, by default, ontologically enriched.

Lastly, this table helps corroborate the previous affirmation regarding the nature of ontological revision in the design process. The Point type is arguably the most basic 3D structure one can employ in design; similarly, the Mesh type is the least complex volumetric representation. These two types alone account for over 60% of the ontologically enriched objects by volume, confirming the bias against deeply nested data structures, and for simplicity, in communicative design exchanges.

### 4.4 Managing Ontological Diversity

One of the major limitations of the composable object model discussed in Sections 4.1 and 4.2 is its limited usability outside a programmatic context, such as either code, scripts or a visual programming environment. In Grasshopper, or a script, the end user can control and define, or map, the relationship between object types and their intended native manifestation in the design software. Nevertheless, in a much more common scenario, where not only does the end user not possess programming knowledge, and, moreover, the only tools at hand are the host CAD application’s user interface, defining the conversion logic between schemas is not possible.

So far, the analysis from this chapter has assumed the end user to be both the producer and the consumer of new object ontologies. Nevertheless, because of the difference of skills involved between operating in a programmatic context and the “standard” one, a differentiation between two categories of users is needed: first, the developer of a schema, and second, the consumer of a schema. While the latter is a skilled technician, he does not possess the programming knowledge needed for defining translation routes between object ontologies that the former has. Gantt and Nardi highlighted this dynamic in their paper “Gardeners and gurus: patterns of cooperation among CAD users“ (1992) and studied how “local software developers” package the code they write for the consumption of their colleagues and the distribution within their companies. Consequently, when defining object ontologies, a developer must be able to, if so needed or desired, package them with their “translation” rules for the various applications in use throughout the AEC industry without regard of, or at least minimising, the limitations of the specific software and skills of the end-user. These concerns, coupled with the technical affordances of the .NET platform, have defined the overall software architecture that underpins Speckle’s approach to managing ontological diversity.

#### 4.4.1 Architecture

To formalise this approach, Speckle coined the idea of a “SpeckleKit”: a packaged
object model with a set of translation routines to and from the software applications it is meant to support. In a metaphorical sense, a “SpeckleKit”, can be seen as a glossary containing a set of conceptual definitions, as well as several independent dictionaries that “explain”, or translate, said concepts in various different languages (Figure 21). Specifically, one can examine the CoreGeometry kit, which groups together the basic geometry types and primitives that have been listed previously in the 4.1 Composable Data Structures section: It consists of the following separate sections:

- A set of class definitions (“the glossary”, containing definitions for the Boolean, Number, String, Interval, Interval2d, Point, Vector, Plane, Line, Circle, Arc, Ellipse, Polyline, Curve, Polycurve, Box, Mesh, Extrusion, Brep objects),
- A partial set of conversions methods for Dynamo (“a dictionary for the dynamo language”),
- A partial set of conversions methods for Revit (“a dictionary for the Revit language”),
- A full set of conversions methods for Rhino and Grasshopper (“a dictionary for the Rhino language”)

![Figure 21: Table view of an imaginary object model kit, which shows the demarcation between the class definitions and their actual conversion implementations in separate CAD software applications.](image)

Multiple kits can coexist and build on top of each other. At the time of writing, two more kits have been developed independently of each other: the first, “SpeckleElements”\(^{21}\) adds a set of basic higher-level architectural objects, namely Wall, Floor, Beam, Column, Level, and Grid. As its geometric base, it uses the CoreGeometry kit, and presently contains translation logic only for the Revit software application. The second, “SpeckleStructures”, is a closed-source kit developed internally within Arup, an engineering and consultancy company, and adds a set of schemas pertaining to the domain of structural engineering (e.g., 1dMember, 2dMember, LoadCase, etc.) and translation logic from Revit, GSA and ETABS\(^{22}\).

Available kits are loaded dynamically, at run-time, by the CAD application plugins in their execution domain (AppDomain). Thereafter, by leveraging the relevant kit “dictionary”, this allows for the automatic conversion from the native application object model to the one defined in a kit. Similarly, it allows the opposite conversion, from the kit’s defined schema to the native object model. If overlapping kits are present, the current default behaviour is to use the first one loaded, or defer this choice to the end-user.

### 4.4.2 Conversion Flow

In order to enact the translation between a host application’s object model and the ones present in the various kits, two methods are present: serialise and deserialise. Similarly, the “dictionaries” of a kit must contain, in order to be complete, two methods for each schema they support: the first, by convention named ToSpeckle, encapsulates the logic that takes a native object and translates it into its counterpart present in the kit; the second, again by convention named ToNative, encapsulates the reversed logic. For example, the signature of such a set of functions looks like this:

```csharp
// Extension method for app to kit object conversion
public static KitObjectA ToSpeckle( this NativeObjectA myNativeObject ) { /*…*/ };

// Extension method for kit object to app conversion
public static NativeObjectA ToNative( this KitObjectA myKitObject ) { /*…*/ };
```

Figure 22: (Code Block) Signature of the ToNative and ToSpeckle functions.

The role of the serialise and deserialise methods is to act as a single entry point for object conversion between object models. Specifically, their role is to efficiently search for and match their given input objects to the correct serialisation routines present in


\(^{22}\) GSA and ETABS are structural analysis software.
the available kits. For example, when the `serialize` function is called on a given object, it will first investigate its context (the kits loaded in its current AppDomain) and check if the object has a direct conversion method defined (`ToSpeckle`) in its relevant kit dictionary. If one is present, it will invoke it using reflection, and return the result. Furthermore, these `ToSpeckle` methods are cached in memory for the duration of the session in order to bypass searching for them again. Nevertheless, if it is unable to find a valid translation method, it will attempt to create a `SpeckleAbstract` object by reflecting on all the properties that the original object, as described in the previous section (4.2 Encoding Existing Ontologies).

The `deserialise` routine follows a similar logic path with the serialisation routine presented above (see the figure below). Upon receiving an object, it will search its existing context for a direct conversion routine for that specific object type (`ToNative` method). If one is found, it will invoke it and finally return the result of the invocation. If no `ToNative` methods are found, it proceeds to look if the type of the given object is `SpeckleAbstract`. If it is not, there is nothing it can attempt to do further; as such it returns the original object, unchanged. This can be considered a failed attempt at deserialisation; nevertheless, no errors are thrown as there are contexts where this behaviour can be considered valid. Otherwise, if the given object is of the `SpeckleAbstract` type, it will proceed to try and deserialise this to its original type.

### 4.4.3 Symmetry

One of the limitations of this approach is the fact that one cannot assume identical contexts across the design process: different actors and stakeholders may or may not be in possession of the kits (schemas and their implementations), or a specific object type has no representation in another application. The former case is possible because, for example, some companies may choose to not share a kit that they developed – as is the case with SpeckleStructures, developed by Arup and not distributed externally. The latter setting is also a common occurrence: while the `Wall` object may exist in Revit and other drafting software, it does not exist in Rhino or other modelling applications, or, for example, NURBS surfaces, while present in Rhino, do not have an equivalent in Blender, or SketchUp. Consequently, an object created in one context may fail to be understood in another one.

Specifically, there are three possible scenarios: given two different contexts, A and B:

1. **Fully shared context**: A and B contain the same kits,
2. **Partially shared context**: A and B share only some kits, but not all,
3. **No shared context**: A and B do not share any kits.
In the case of (1), an object originating from context A will be understood in B. In other words, the `serialise` and `deserialise` functions are symmetric:

```plaintext
myObject == deserialise( serialise( myObject ) ) // evaluates to true
```

In the case of (2) and (3), the above statement evaluates to `false`. In the case of (2), the `deserialise` method will attempt to degrade gracefully. Concretely, it will look for the types that a given object inherits from in reverse order of abstraction, and for each of those types tries to find a `ToNative` method valid in its given context. This approach allows for a “lossy” interoperability process, and is particularly useful in many scenarios. For example, when translating between Revit and Rhino a `wall` object, even if Rhino does not have a native object supporting such an object, the `deserialise` method will be able to return a `mesh` object with several converted sub-properties, such as base line and thickness, specifically because, in this example, `wall` inherits from `mesh`. This enables various stakeholders to partially understand each other’s data, and thus enables their coordination.

Summing up, one must not assume that the `serialise` and `deserialise` routines are symmetric. Similarly to the popular whispers (apocryphally Chinese) game, the process of ontological revision introduces a certain mutability in the flow of data. This has an important bearing on the issue of consistency, and specifically object identity: if an object changes by being interpreted in different contexts, how can one avoid the issue of data and fidelity loss? The way this limitation is mitigated is presented in Chapter 5, Data Classification, where an important feature of the whole communication framework, namely unique hashing and immutability, is introduced.

### 4.5 Conclusion: Enabling Ontological Diversity

The proliferation of informal standards may be seen as detrimental to the consistency required in the digital design and construction process. Nevertheless, so far, consistency has been wrongly equated with rigidity, by virtue of the fact that existing formal approaches rely almost exclusively on the IFC standard, whose stated aim is to be the single source of schemas describing all of the built environment’s constituent objects.

This chapter has introduced a new approach to managing and specifying digital data representation, one that is primarily emphasising composability and that aims to enable, rather than dissuade, the act of ontological revision that various stakeholders must embark upon in order to streamline their communicative processes.
Followingly, this approach has been evaluated in three different contexts. First, Section 4.1, Composable Data Structures looked at how end-users created and evolved their own ontologies, or “ad-hoc” object models, for the purpose of exchanging information across disciplinary boundaries. Second, the affordances of programmatically matching the composability of the lower-level approach with existing, higher-level standards has been evaluated in Section 4.2, Encoding Existing Ontologies. Third, in Section 4.3, Managing Ontological Diversity, the investigation focused on what would be the best way to enable multiple object models within a digital design communication process. Succinctly put, the findings can be summarised as follows:

- A lower-level, composable object model enables a productive process of ontological (representational) revision, as evidenced in Section 4.1.2. Section 4.3 reveals that 18% of the total objects transferred by Hestia, the test Speckle server, have been enriched with custom structures by end-users.

- Existing higher-level object models from the industry, such as BHoM (Section 4.2.2), as well as IFC (Section 4.2.3), can be natively supported, thus allowing for “backwards-compatibility”.

- Further, Section 4.3 reveals that end-user driven ontological revision tends to minimise the complexity of the data structures involved (an average tree depth of 1.97, as opposed to 3.43 for pre-existing schemas). Moreover, there is strong evidence as to the preference for base “geometric atoms” when custom schemas are articulated. This also results in a leaner communication volume, as average user-created object sizes are approximatively seven times smaller than ones derived from existing ontologies (1kb vs 7kb), and four times smaller than the overall average (1kb vs 4.5kb).

- The reduced complexity of data structures allows for more user inference to manifest at the receiving end, and subsequently requires more user intent from the sender.

- The programmatic tests from Section 4.4 demonstrate that multiple, self-contained object-models can be programmatically supported in a simultaneous and consistent manner in a digitally enabled design communication process, thus invalidating the industry’s assumed need for a singular, unique object model.

In short, the ontological richness and diversity of the AEC industries, rather than being suppressed, can be embraced and supported in a productive way by current technical affordances. As Sperber and Wilson remark about natural (social) communication processes (Wilson and Sperber, 2008), information exchange in the
digital design context tends to follow a path of least resistance by maximising the relevance of the input and minimising its complexity.

Lastly, by being codified and decoded in different contexts, information changes throughout the process of communication. The approach described in this chapter suffers from a similar limitation: because one cannot provide a guarantee as to the availability of shared object models between these domains, one cannot guarantee a symmetrical behaviour of the codification and de-codification functions (serialisation, deserialisation). This is a major implication that will be specifically tackled by the following chapter, *Data Classification*. 
5. Data Classification

or, a Programmatic Base for an Object-Based Persistence Layer in AEC
5.1 Challenging Centralisation

The previous chapter introduced a new methodology for data encoding and decoding that was aimed at better suiting the AEC industry’s communication needs. Going forward, this chapter will investigate what is the best way to persist (or store) the information that is being transacted within the design process by comparing existing methods, which are largely model- or file-based, to an alternative approach that is built on top of the solution proposed in Chapter 4, *Data Representation*.

Traditionally, digital design data is stored within models. It is mostly produced in specialised modelling or drafting software and thereafter saved in files on a local hard drive. The concept of a file dates back to the 1950s, and its definition has remained, since then, rather stable: it is an ordered array of binary information that can be, in one way or the other, decoded. These decoding algorithms are, depending on the file itself, open source or proprietary. In the case of AEC software, most of it is reliant on the latter kind of software; as such, one needs to have access to the specialised modelling software that produced it in the first place, or one that has a similar decoding capability as well. Subsequently, design files can be seen of having a certain atomicity and impregnability; they resist outside scrutiny; this has tended to isolate the information produced by the AEC industries from the outside world.

The *de facto* collaborative approach in AEC is centred around the idea of a federated, central model, or one single source of truth, that contains, synchronises and validates the information produced by all technical stakeholders involved in the design process. This can be traced back to Eastman’s work on GLIDE (Eastman et al., 1975) and BDS (Eastman, 1980). Essentially, it mandates that all the discipline specific models that are being produced within the design process are centralised within one “master” model, which represents the single source of truth. It evolved as an answer to the perceived “babelian” situation of the AEC industry (Hamil, 1994), in which all the stakeholders would talk to each other, apparently at random and without the possibility to coordinate altogether—see the left side of the diagram below (Figure 23). The proposed solution was twofold: first, a centralised common standard for data representation, namely IFC, which was discussed at length in the previous chapter; second, a centralised model that would hold all the information in one specific place and be the vehicle through which all communicative exchanges involving design data would take place; that is, the carrier for the data’s representation.
The “one model” approach is reflected in the mandatory inter-relation of building objects prescribed by the data representation standard in use or by the architecture of the software used. For example, as mentioned in the previous chapter, *Data Representation*, the latest IFC release states that “round-tripping” of single objects is out of scope and, in general, not supported (Bentley Systems, 2008; buildingSmart, 2016; Thein, 2015). This is because to “round-trip” an object one needs to extract it from its context, and be able to decode it meaningfully without access to the file it originated from. Moreover, the previous chapter has also shown how encoding an object defined by a canonical implementation of the IFC standard actually implies serialising all the other information present in the IFC file itself, including any other objects, because of the tight coupling of the class definitions.

The in Figure 24 is also used to describe the mandated workflow practice for exchanging information—communicating—in a design process, between all the actors involved. Informal communication practices, for example ones bypassing the central model, are discouraged: there is an assumed equivalence between the technical model and the social one. Nevertheless, *Chapter 2, Literature Review* has shown that appropriating technical models of communication and mapping them on
other, non-technical domains, may be detrimental to the process they are attempting to describe or enable. Consequently, the following chapter can be seen as challenging the current projection of a technical model of information exchange onto an essentially psycho-social phenomenon of communication.

Further supporting this challenge is the fact that practice, as well as literature, show a proliferation of customised workflows. These are being developed internally by various AEC companies to serve various project specific (or company specific) interoperability, communication and data processing needs. Summarily put, these efforts amount to extracting, filtering and curating data out of the monolithic files that digital design information usually resides in, and re-assembling it for different purposes, usually dependent or defined by various actors that are involved in the design process. These customised workflows can be seen as informal communication channels—usually, they are not legally specified or part of official documents—that evolved from the communicative interaction between various groups of actors. As such, freed from the weight of an official imposed methodology, they usually are lean and efficient: the parties involved have had the space to minimise cognitive as well as machine processing effort, as well as optimise the quantity of information being transacted. Consequently, the following chapter will assess the viability of an informational system for the design process that can support the emergence and evolution of a natural communication network between the actors involved in the design process.

5.2 From Sharing Files to Curating Data

In order to understand what is the best way to structure design data information for communicative purposes, one has to first look at how it is produced, or created, within the specialist software it originates from, as well as how one can expect end-users to structure it. The following section shall assess the process of classifying information in existing CAD software and, most importantly, how different interactional mechanisms of sharing influence the nature and quality of the communicative process.

Traditional modelling applications produce design data through direct user input: an architect, engineer or other specialist actor draws building elements. Afterwards, users select objects with their mouse in order to interact with them, and when they do so they either choose one object or more; in the realm of parametric modelling, the

---

23 These methodologies are described as “informal” only because of the self-definition of a different approach as being the “formal” one.
output of a given function—either a singular object or an ordered series thereof—is then selected by the user as the input of another operation. Nevertheless, a file-based sharing approach essentially eliminates the cognitive step of “selection”, or filtering, at an object level; the whole model is “communicated”. Consequently, when centralising all information in one model as current formal practices mandate, the information thus exchanged is dissociated from its meaning and reason to exist, as there was no prior step of selecting what is needed or for what purpose. The designer “shared the model”—but why, and for what reason? Unfortunately, this results in informational waste that adds friction to the communication process: it places a big burden on the receiver to filter and extract what he or she needs from the information available to them, not to mention the technical implications that arise from transacting, storing and managing the large quantities of design data that are produced.

Thus, a communication process that is based on purely passive production of information without heeding its relevance or purpose can be seen as dysfunctional. It violates Grice’s third maxim of the “Cooperative Principle”, namely the one that simply states “be relevant”, as well as the first maxim, that of quantity (“make your contribution as informative as it is required; do not make your contribution more informative than it is required”) (Grice, 1991). Furthermore, this sharing model does not encourage end-users to maximise the communicative relevance of information, as described by Sperber and Wilson (2012), because it increases processing effort required on the behalf of the receiver, and lowers the potential for positive cognitive effects to emerge.

To exemplify, the diagram below is one of the results from a workshop on practice workflows undertaken together with this project’s industry partners (Figure 25). Data from this workshop serves as the basis of two aspects of the analysis: first, it represents an inquiry into how data is structured in AEC data flows; second, it serves as a basis for the evaluation of existing transactional friction thereof, which will be discussed at length in the following chapter. Looking through the former lens, it highlights a recurring pattern of information distribution in one of the most common workflows in the design process, namely that between an architectural stakeholder and an engineer. It reveals the fact that, in order to perform the required analyses for a building project, an engineer typically curates the architectural model, with its geometry and associated metadata, in several different ways. These curated sets of data allow her or him to perform specialised tasks using digital tools specifically geared towards solving subsets of engineering problems, such as seismic analysis, wind analysis, footfall and crowd simulations, energy studies, etc.
Figure 25: Diagram showing the typical ways design information is cast into separate models for specific analysis tasks, and the subsequent feedback of the results in the design model.

For example, a wind analysis requires, in most cases, just the outer shell of the building – its façade. An energy analysis model is based on volumetric room information coupled with the openings (windows, doors, etc.) and their nature (glazing coefficients, thermal inertia, and other material properties). One of the most computationally intensive analysis, the seismic assessment of a structure, usually takes as an input a three dimensional volumetric model of the structure, which thereafter it transforms into a four dimensional model (including the “inside” of objects). All of these models, at their base, share a common model coming from the architect, nevertheless they each require specific subsets of data from it, and apply custom transformations on it. Currently, the act of curating the right data for the right task is happening through a tedious process of generating secondary models from a “master model”, typically provided by the architect. These are thereafter exported as separated files and subsequently imported in the specialised analysis software. The participants revealed that, in some cases, they can spend up to half of their day cleaning, curating, exporting and managing this specific aspect of the process.

This situation arises because the effort of “decoding” information, in a file-based collaboration environment, is left entirely to the receiving party; the sender just “shares the file”. Subsequently, the proposed alternative approach seeks to balance
the effort of curating information. Subsequently, it requires a cognitive action from the communicant of pre-filtering, or selecting, the objects that will be subsequently encoded, persisted in the database, and thus available to others. By doing so, actors involved in the design process are encouraged to “think before they speak” – the extent to which this approach was successful is analysed later on in this chapter, in Section 5.7.1. In other words, this amounts to assessing and shaping the relevancy of their communicative act: instead of simply sharing all the design data they produce (and thus place a high processing barrier on the receiver as well as risk communicating the wrong data), they must engage in a cognitive process of evaluating what information to share and for whom it would be relevant.

When design information is produced, usually the software employed has its own specific ways to classify information in order to help the user in achieving their task. Excel, for example, is based on a tabular paradigm of rows and columns; nevertheless, CAD applications usually have multiple ways of organising design objects. These can be either based on a hierarchical scene graph, in the case of CATIA, or simpler systems such as layers, which are present in Rhino and AutoCAD. Other applications, for example Vectorworks, provide more ways to classify data: objects are categorised based on “classes”, “layers” and “levels”. Furthermore, most software allows for objects to be grouped together in either “Groups”, which are just a superficial way to keep them together, or “Blocks” which are archetypal representations of one or more objects that can have multiple instances existing with different transform properties. Most probably, there are many other conventions for how design data is structured depending on the software used. Furthermore, visual programming introduces a separate, programmatic (code-based), approach to creating design data in which the produced information is ordered and grouped by the functional operation that created it.

At a programmatic level, the most productive common denominator amongst all of the models above, when it comes to information sharing, is a list of objects. In other words, a list is a group of objects that an end-user would want to transmit to another. This group may be ordered, if produced in a programmatic environment, such as a parametric modeller, or unordered if the result of an arbitrary selection. Moreover, a design model is essentially a group of objects; all the ways of classifying these objects (layers, classes, levels, etc.) are essentially customised ways of filtering and structuring this object collection based on an element’s properties (which can be either inferred or directly assigned to by the designer). Furthermore, design information is in continuous flux as it is produced and revised throughout the process. As such, an initially shared selection will change through adding, editing and/or removing objects.
Consequently, in order to facilitate the end-user to curate and share dynamic groups of objects, a new object is needed, namely a SpeckleStream. Essentially, it represents a group of other SpeckleObjects that have been selected by a communicant in order to be shared. It can be as large as a whole model, or as small as to contain only one object. Its naming implies its dynamic nature: SpeckleStreams can be constantly changing, nevertheless they exist independently of the objects they collate and thus provide a fixed point of reference that can be later on referred to. The following section expounds on the technical implementation needed to materialise this concept.

5.3 Enabling an Object Oriented Data Platform

The previous chapter has described the specification of an extensible low-level object model for digital design data, as well as its implementation in C#, based on the .NET framework. This implementation allowed for the serialisation and deserialisation of objects—that is, encoding and decoding them and is part of what usually is called the application layer. As follows, it needs to be extended to incorporate a persistence layer—or memory—where these objects will be stored, retrieved from, and classified.

5.3.1 System Architecture

The approach used is following a three-layer architecture pattern, which is a classical and wide-spread multi-tier software design approach (Gamma et al., 2015). Essentially, this means that an information system is segregated into distinct parts that deal with one or more specific application concerns independently of each other. These application compartments interact through clearly defined rules, usually encapsulated in the form of APIs. The three main layers are usually referred to as the (1) presentation layer, the (2) application layer and the (3) persistence (or storage) layer. (1) is usually the topmost level of the application, and is a layer users can interact with directly (for example, the GUI); (2) encapsulates all the logic that the application requires, and is usually separated as well into multiple sub-layers; finally, (3) contains the persistence mechanisms, such as databases or file shares, where an information system can persist its state (Gamma et al., 2015).

The most direct approach is to write the serialised object data directly to disk as a binary blob and subsequently rely on a file share to make this data more widely

24 During the course of this thesis, users can be either (1) developers that build on top and extend the functionality provided by the research instrumentation, or (2) end-users, such as designers, engineers or architects who interact with the system as a “finished product”. In this case, the text is referring to the latter.
accessible; nevertheless, this technique would suffer from the same resistance to scrutiny that was pointed out in the beginning of this chapter, as it is reliant on files. Consequently, the technical instrumentation, Speckle, shall employ an open source document-oriented database, MongoDB, as its storage layer. The motivation as to this specific choice—there are other similar solutions, such as RethinkDB, CouchDB, etc.—lies in the fact that it is a well-documented and flexible database that allows for the coexistence of structured as well as unstructured data—fact which matches with the open-ended schema definitions presented in the previous chapter. The option for a NoSQL\(^{25}\) instead of an SQL solution (for example, MySQL or MariaSQL) was driven by the fact that Speckle will need to be able to query the memory layer also based on non-indexed fields—basically on the ad-hoc defined properties of composed objects.

As usual, a given storage layer exposes an API through which data can be written into the system and as well read out. Therefore, a potential approach would be to extend the .NET-based `SpeckleCore` library with client read-write capabilities directly to the MongoDB database. Nevertheless, a middle application layer is introduced. Its role is to mediate the transactions between the database and the clients that consume and produce the data and offers room to enable various system-wide cross-cutting concerns, such as user authentication, permissions, data validation, rate limiting, security and encryption, and a more universal way of exposing data to outside actors and third-party applications. This middle application layer is based on a REST API and is implemented through a NodeJs based server (`SpeckleServer`). The `SpeckleCore` library is extended with an `ApiClient` class that exposes methods for calling the `SpeckleServer`.

The diagram below shows a schematic view of the Speckle ecosystem (Figure 26). An end application, such as Rhino or Dynamo, in order to communicate with a `SpeckleServer`, will incorporate a relevant API client. Specifically, in this case, `SpeckleCore`, as it is based on the .NET platform, same as the example host applications above. This API client will, subsequently, call the server’s REST API endpoints, which are described in more detail in the following section. Herein, calling entails passing any data required to perform the given operation. Subsequently, the server parses and validates the data it received. If everything is in order, it is persisted to (or retrieved from) the storage layer, in this case MongoDB (discussed in the following section, namely 5.3.3).

---

\(^{25}\) SQL stands for Structured Query Language. NoSQL does not imply “non-structured” query language—this is a common misconception; the acronym actually stands for Not Only SQL.
It is important to note that, from a web-centric view, Speckle can be divided into the “Backend”, consisting of the SpeckleServer and the database, and the “Frontend”, consisting of various end-user facing applications (e.g., CAD plugins or web apps). The application layers presented above can be deployed on different hardware, independently of each other (n-tier architecture) or one single machine (tier-1). Further to be discussed in Section 7.4 of the Discussion chapter, while a given instance of the SpeckleServer is a locally central entity, because it can be deployed multiple times by multiple actors, at a higher abstraction level, it is a node in a decentralised network. This is because the clients are not restricted in their interactions to one given SpeckleServer at a time and can connect to multiple ones simultaneously, essentially acting as bridges between instances (Figure 27).
5.3.2 API Specification

The Speckle Server’s REST API was documented (specified) using the OpenAPI v.2.0 (Swagger) specification language. The remit of the specification encompasses the platform-agnostic definitions of a set of REST endpoints for the basic CRUD (Create, Read, Update, Delete – the four basic functions of persistent storage) actions that the SpeckleServer subsequently implements. These actions will be called by the various front-end application clients, such as the one described in the following section. The code sample below shows how such an action is defined, namely the `ObjectCreate` endpoint. Specifically, it defines at what URL endpoint the server will accept calls for this action (`/objects`), what kind of action this is, via the HTTP verb (“post”), an operation id `ObjectCreate` by which one can refer to it consistently throughout the various parts of the ecosystem, operation parameters (in this specific case, an array of SpeckleObjects), and a set of possible responses (“200” for a successful operation, “400” for an error).

```javascript
const winston = require( '../config/logger' )
const BulkObjectSave = require( '../middleware/BulkObjectSave' )

module.exports = ( req, res ) => {
  if ( !req.body ) {
    res.status( 400 )
    return res.send( { success: false, message: 'Malformed request.' } )
  }

  BulkObjectSave( req.body instanceof Array ? req.body : [ req.body ], req.user )
  .then( objects => {
    res.send( { success: true, message: 'Saved objects to database.', resources: objects.map( o => { return { type: 'Placeholder', _id: o._id } } ) } )
  })
  .catch( err => {
    winston.error( JSON.stringify( err )
    res.status( 400 )
    res.send( { success: false, message: err.toString() } )
  })
}
```

*Figure 28: Implementation of the CreateObject POST endpoint.*
The above describes the “Create” action (Figures 28 and 29); the “Read”, “Update” and “Delete” actions are similarly specified. The full SpeckleSpecs contain several more groups of endpoints for actions pertaining to creating accounts, projects, and comments (that are not covered in this thesis). Nevertheless, the following section shall discuss in more detail the database structure and introduce a new schema for a collection of SpeckleObjects, namely a SpeckleStream, which will serve as the basis for a convenient flat logical grouping of design objects that is easy to reason about.

5.3.3 Database Model

The MongoDB database system, as it is, can store object information without regard to their structure. Nevertheless, this approach does not scale well after a certain amount of data and/or users, as the efficiency of later on querying the system is reduced drastically on fully unstructured collections. Consequently, Speckle employs a structured approach that attempts to strike a balance between flexibility and rigour.

Subsequently, the database is structured in several collections (or tables, as they are known in SQL), the relevant ones for this discussion being DataStreams, SpeckleObjects, Users, and UserClients. Each collection will hold only its specific type of object, and
perform minimal data validation to ensure the consistency of the persisted information. Particularly of interest is the relationship between DataStreams and SpeckleObjects: the former collection stores exclusively objects of the SpeckleStream type; the latter holds, essentially, all the SpeckleObjects themselves.

A SpeckleStream’s schema specifies an array named “objects”, which stores a list of database ids that reference objects in the SpeckleObjects collection. This approach allows for one object to exist in multiple streams, which is particularly useful in scenarios in which a user shares two overlapping sets of design objects. The UserClients collection keeps track of objects that define where a SpeckleStream originates from and where it is subsequently received; they will be discussed in more detail in the following chapter. The diagram below describes the relationships between the various collections (Figure 30).

*Figure 30: The Speckle database model, showing a simplified view of the collections (tables) and relationship between them.*

The actual data model is more complex than the one presented above, nevertheless for the purposes of illustrating the basic functionality of Speckle it suffices. An important factor in the design of an object-oriented database is defining the indexes of a collection, as this is the primary mechanism by which data is retrieved, sorted and classified. Consequently, the SpeckleObjects collection is indexed by their primary _id, their hash as well as by their type. The SpeckleStreams collection is indexed by its shortId field (which is a unique five letter and number key provided for ease of
reference), as well as by its owner. These additional indexes allow for queries based on them to operate at the same level of efficiency as they would in a traditional SQL database, without a complete table re-scan.

As the previous chapter has shown, a SpeckleObject may be composed of many other different sub-SpeckleObjects. There are two different approaches that can be taken in this scenario regarding the way they are saved in the database. The first is to ignore their composition and simply save it as one document that encompasses all other documents. The second approach is to recursively unpack all the sub-objects and save them as separate documents, and re-link them based on their database references or unique hashes. The second approach is more efficient in the long term with regards to the amount of storage resources. Nevertheless, it places an extra burden on the server when it comes to retrieving information, as the number of reads necessary grows linearly with the object’s composition depth (each sub-object needs to be read individually from the database and recomposed into the parent object). During this research project and the evolution of Speckle, both approaches to object storage have been implemented, nevertheless only the former remains in active use due to constraints on both developer usability as well as code maintainability.

5.3.4 The Speckle Server

The API is implemented as a stateless multi-threaded Node.js server that respects the contract embodied by the SpeckleSpecs. Beyond providing the means to create, read, update and delete SpeckleObjects, the server application also handles various security concerns, such as encryption, and, most importantly, user accounts and permissions. A detailed discussion of these aspects is beyond the current scope of this investigation.

The technological stack used to implement the REST API relies on the Express framework. This is used to define the URLs at which software clients can thereafter call the specific methods, for example POST /api/v1/objects is used for saving into the database one or more SpeckleObjects. Specific idempotent handling functions for these operations are subsequently written as separate application controller files. The authentication of end-users is handled via the Passport module via the standard, well used and tested strategy of username (email) + password26.

Previously, it was mentioned that having a stateless server is important for infrastructural scaling purposes; this is achieved through the use of expiring JSON

26 Currently, the Speckle server implements several other authentication strategies that enable it to integrate with existing corporate environments (Auth0, Active Directory, Azure AD, Twitter, Github).
Web Tokens (JWT) which are provided to clients upon a successful authentication. For communication with the MongoDB database, Speckle relies on a higher-level driver, namely *mongoose*, that simplifies some of the operations involved as compared to the native NodeJs driver.

### 5.3.5 Application Integrations

The *SpeckleCore* library is subsequently extended with a series of methods that can call the API exposed by the *SpeckleServer*; these methods are grouped with a *SpeckleApiClient* partial class which is generated directly from the *SpeckleSpecs* using, as in the previous chapter, NSwag (Suter, 2018). These methods allow for the interaction between the client application with the persistence layer (a MongoDB database), mediated through the *SpeckleServer*. The *SpeckleApiClient* class is further extended to add several convinence methods that inject in every server request the user’s authentication token, if one is present, as well as compress the informational payloads being sent to server.

Consequently, the application integration between Speckle, Rhino and Grasshopper is extended to allow users to persist design data, as well as receive it. The Grasshopper integration consists of two components, namely the *SpeckleSender* and the *SpeckleReceiver* (Figure 31). The sender component, when instantiated, creates a new *SpeckleStream* that is subsequently populated with the objects that a user connects to it. The receiver component, once provided with a SpeckleStream’s *shortId* property, will retrieve all the objects thereby present from the database. Whenever the input of a *SpeckleSender* changes, the objects are saved to the persistence layer and subsequently the *SpeckleStream* is updated automatically.

![Figure 31: Grasshopper Speckle Sender (left) and Speckle Receiver (right).](image)

While in the programmatic environment offered by Grasshopper the approach was
easily determined based on already existing user interface constraints, the direct to Rhino integration required special handling. Together with Luis Fraguada from McNeel Europe (one of the InnoChain industry partners), the author has developed a basic user interface that exposes the send and receive functionality in a quasi-similar way to the already present “layers” interface present in Rhino (Figure 32). It allows users to create SpeckleStreams from an initial selection of objects from the file currently being open, as well as adding or removing other objects later on. Similarly, as the Grasshopper clients, it tracks the objects assigned, and, if changes are detected, it will automatically update the SpeckleStream’s state.

![Image](image.png)

**Figure 32: Rhino Sender (dark blue) and Receiver (teal).**

The SpeckleRhino interface is functioning independent of Rhino itself and communicates via a special class that handles passing messages between the two layers. This allows for its reuse in other applications, for example Revit. The reasoning behind this architectural and technological implementation is anchored in the need to provide a consistent user experience across multiple software packages, as well as other technical concerns, such as cross-platform compatibility.

The automatic detection of application state changes and subsequent updating of the database can be disabled (paused), or further refined to enable different interactional possibilities for end users. For example, instead of automatically sending updates, one could require a direct user action, such as a button click, and a description message to be written by the user that explains or justifies the changes being made, as in the case with source code version control software.
5.4 Object Identity, Immutability & Data Deduplication

The previous chapter showed that the inferential approach to design data encoding and decoding results in a process that is not symmetric, or in other words, information is changed throughout the process of communication. At a superficial glance, one would conclude that it is therefore impossible to maintain consistency, or a pertinent state of truth within such a system. Nevertheless, the following section will show how one can provide a consistent and rigorous system that bypasses this issue by ensuring the immutability of SpeckleObjects at the storage layer and, at the implementation’s programmatic level, not allowing simultaneous edits from two different sources of the same object collection (SpeckleStream).

Immutability is the property of an object whose state cannot be changed after it has been created. A corollary formulation can be seen as if one modifies the properties of an object, it becomes a new object. Consequently, a new way of storing design data that takes this into account is put forward: whenever a user persists new objects to the database, old ones are never overwritten. This describes an “only forward” approach. In order to do so, one needs a way of efficiently checking whether an object changed and is now a new one, or it’s still the same; in other words, Speckle needs a consistent way to uniquely identify objects.

When stored in a database, each object gets assigned a unique identification. These are stored under the _id field and serve as primary indexes for that collection. Similarly, at the application layer, each object has a unique address or reference pointing at a specific location in the stack or, in the case of SpeckleObjects, the heap where its information is stored. Nevertheless, these unique identifier keys are assigned randomly by the system at hand and do not reflect on the actual properties of the object itself; as such, there is no way quick of knowing whether one object is the same as another without comparing its properties and fields one by one. In order for this approach to be viable, a faster and easier way is required to ascertain and track object identity in order to ensure their immutability.

However, the object’s hash provides an efficient way to ensure a consistent, non-random and immutable approach. As discussed in Section 4.1, Composable Data Structures, each SpeckleObject has to have a unique hash (digest) that is directly dependent on its value and should change whenever any of its properties change. For example, a simple point object with its coordinates at (0, 0, 0) will have a different
hash value than one with its coordinates at \((0, 1, 0)\). There is a secondary hash, namely the \texttt{geometryHash}, which operates the same way, but its value is not dependent on a \texttt{SpeckleObject}'s custom user added fields. This allows us to later on \texttt{diff} objects at two levels, a holistic one, and a purely geometry-information based one, which enables certain optimisations with regards to the size of a transaction. Subsequently, one can ensure that information loss does not occur: even if the information transmitted through the system is being changed by the contexts in which it is received, one will always have a record of the state it originated from.

Furthermore, this has the benefit of allowing us to not store duplicate objects. For example, consider designer Mary using a large site model for a building project that was made previously by the surveyors. If she will save this model to the database, assuming it is only one object, it will subsequently be stored under its unique hash. John, Mary’s colleague, uses the same model, and will save it as well in Speckle’s database. Nevertheless, because the site model’s objects hashes are already present, the \texttt{SpeckleServer} that mediates these interactions will detect and prevent saving them twice. This amounts to data deduplication based on object fingerprinting (their hashes) and greatly increases the efficiency of the framework in dealing with system resources.

As a technical implementation detail, this same mechanism is employed on the client side through the use of a local cache that keeps track of what objects have been previously successfully persisted on a given Speckle server. This allows to pre-empt the transaction between a client and the server of a previously persisted object: instead of sending the duplicate to the server, the client can already cull it beforehand and only pass its reference if needed.

The behavioural picture of the system is completed by discouraging the simultaneous editing of \texttt{SpeckleStreams} (object collections). Essentially, one can update a stream only from the original context or source it has been created. Nevertheless, this rule is not enforced at programmatic level within the core application layers—it is merely a convention of the end-user facing plugins of the instrumentation. At the receiving end, if one needs to modify the information received and send it back to the other communicant, one first has to pull the associated objects into the host application. After modifying the objects, the end-user will create and save a new \texttt{SpeckleStream}

---

27 The issue of continuity – or how one can identify that a modified object is the same, conceptually, as its predecessor – is tackled in Section 5.5, Data History.

28 A \texttt{diff} is a data comparison between data objects through which one calculates and displays the differences and similarities between them.
that contains the modified data. Subsequently, this new SpeckleStream can be received by the original party. It is important to note that, whenever an end-user creates a SpeckleStream, it is stored in the SpeckleServer’s central database, together with its constituent SpeckleObjects. These can be subsequently received by other end-users and not just one single recipient, leading to the creation of a “human”-based directed graph. The diagram below shows a schematic view of the process described above, first from the point of view of only two users exchanging information through two streams (Figure 33), and thereafter expanding to incorporate multiple streams and actors (Figure 34).

Figure 33: Diagram showing a feedback loop between two users working on the same geometry: A, via stream 1, sends to B an object. B modifies the object, and sends to A, via stream 2, her or his proposed changes.
Figure 34: Expanded diagram showing the interactions between multiple actors and SpeckleStreams, and the emergence of a communication network in the shape of a directed graph. Stream A is being received by three different users. These users create new streams, some of which reach back to the original sender and influence the data being sent in Stream A.

While the lack of concurrent editing can be seen as limitation, it is an extremely productive one if respected\(^29\). Productive social dialogue does not happen concurrently, in other words communicants do not all speak at the same time; conversation, as Garfinkel and others put it, implies an ordered sequence of informational transactions that build on top of each other. Speckle, first and foremost, aims to enable the same process at the level of digital infrastructure for design communication. The transactional aspects of design data communication, and how their quantitative qualities, such as transaction size, reflect on its progress and the overall framework architecture, will be analysed in detail in the following chapter.

5.5 Data History

Having a historical record of what information has been communicated is an important requirement that emerged from the interactions with industry. In a file-

\(^29\) As mentioned previously, simultaneous editing of the same object collection is disallowed by convention; programmatically, there is no barrier preventing it from happening, and some users have put it to good use.
centric approach, this is usually a wasteful process that requires keeping multiple copies of the design models being produced throughout time. Besides the logistical difficulties of keeping track of each version, the process does not allow easy comparisons between the stages thus archived. Nevertheless, based on the approach described above, this section describes an efficient way of storing design data state through time. This can happen on two levels, either object-by-object, or object collection by object collection.

5.5.1 Object Level Tracking

Object history is feasible only in cases where one can map a common identity between Speckle’s persistence layer and the application that produces the object itself. As such, this excludes objects that are produced by a parametric model\(^{30}\), either from Grasshopper or Dynamo: by being in continuous flux, they cannot be related to their “past” instance. For example, consider a component that produces a set of random points in a three-dimensional volume based on the desired number and the desired rectangular volume that needs to be populated: each change in the inputs will generate a different output that cannot be related to a previous one, as there is no object-level consistency, only at a generative level one can identify a constant identity (the function itself).

Nevertheless, in the case of objects that are produced by end-users in a non-programmatic manner, history can be traced. For example, consider a mesh, object A, with a given application id\(^{31}\). The designer, after initially modelling it, proceeds to modify it in a certain way, which, when encoded into a SpeckleMesh, will receive a different unique hash, thus being transformed into object B. Nevertheless, the application id of both object A and B is unchanged, allowing us to identify a relationship of kinship between the two.

5.5.2 Stream Level Tracking

Given the fact that object level information is decoupled logically from the object collection itself (the SpeckleStream), storing versions of its state, at a given point in time, becomes computationally cheap. As such, every change in the composition of the SpeckleStream’s object array can be stored and retrieved later, as a child of the original. For example, consider an initial set of three objects, A, B and C, that are being persisted in a given SpeckleStream. The user, through subsequent modelling

\(^{30}\) Not to be confused with parametric modelling software, such as SolidWorks.

\(^{31}\) All applications keep track of the objects in a given file using one form of an id or another. For example, Rhino uses randomly generated GUIDs.
operations, removes object B and replaces it with two other objects, D and E. The original state has a SpeckleStream composed of A, B, and C. After the modelling operations, it will contain A, D, E and C.

Because the SpeckleStream itself keeps track only of the references to objects it is composed of, storing a reference of its historical states does not imply duplicating the data of recurring objects; as described previously, only unique objects are saved in the database. The actual cost of storing a record of a stream’s state at a given point in time amounts to simply storing a new array of object references. If the state is composed only of previously existing objects, it can be negligible.

The approach described herein is just a skeleton on which future functionality can be built. The way the history mechanisms of the storage layer can be exposed is flexible: for example, one can envision a fully version-control like system being developed and consequently implemented in the presentation layer. Another, more pragmatic and easier to implement, way is to simply enable design “versioning”, in other words allowing actors to save the state of a given SpeckleStream as a design variant, or option, without necessarily seeing it as part of the more complex version control tree. To a certain extent, both systems require users to be able to compare the changes between one or more states, which is implemented as a special route on the SpeckleServer's API. The change detection algorithm currently implemented compares the object array of two different SpeckleStreams (say, A and B) and returns three separate lists of objects that represent the intersection of the two sets, as well as the difference between each of them (A ∩ B, A - B and B - A). In other words, they correspond to “objects in both A and B”, “objects only in A” and “objects only in B”. If B was a newer version of A, this operation essentially reveals the unchanged, deleted and added objects.

Further to be discussed in the following chapter, Data Transaction, the same mechanisms are employed also by the client application integrations to optimise transaction payloads, as well as by the server to remove data duplication. It can be seen as a way by which technical communicants—the code itself—leverages the system’s memory of previously persisted data, to the extent it is aware of it, in order to optimise the relevancy of the information being transmitted: the immutability and “only forward” storage layer, that has been introduced previously, when taken together, not only prevent transactional data loss, but simultaneously allow for the coherent aggregation of a historical record of communication.
5.6 Case Studies

As mentioned in the Methodology chapter, the Speckle framework has been continuously exposed to feedback from the industry and the InnoChain project partners throughout its development. Consequently, the approach described in this chapter is a direct result of its usage and specific case studies are difficult to extract as independent experiments. Nevertheless, in the following section, several scenarios are presented that highlight the affordances provided by increased flexibility of design data classification, as specifically implemented in Speckle.

5.6.1 Curating Data

Dialog Design, a large multi-disciplinary design company based in Canada and North America, has been using Speckle at the core of their digital design data communication processes. One of the ways they employed the framework was to enable the integration of simulation early on in their design process. They did this by having an energy analysis tool running on a VDI farm and directly interfacing with the SpeckleServer to provide almost real-time sustainability analysis (under 30 seconds) on a design model. At the core, this integration relies on having two SpeckleStreams, one embedded as a sender in the design model through which the necessary information would be sent to the analysis server, and a second one carrying back the information from the analysis back into the design model.

Dialog Design’s approach relied on curating from an architectural model the specific information needed for performing the analysis. In this case, they sent just the building envelope (outside massing shape) as one independent collection of objects (SpeckleStream). Subsequently, they retrieved the data produced by the environmental analysis software and matched it back with the original model via a separate SpeckleStream. Based on these results, they were able to inform further design decisions on the volumetric and orientation of the building being designed.

A further application of the Speckle framework inside Dialog was in the field of building programming. Building programming is a complex task in which one has to match specified desired surface areas for the various functions that a given building has to incorporate with their geometrical constraints, often expressed in terms of accessibility and connectivity (for example, a requirement can be formulated as the back-offices of a municipality building—300m2—need to have quick access to the archives—2000m2, underground placement). Dialog devised a workflow for optimising this process for existing buildings: (1) from Revit, where the original building model is defined, area plan outlines are selected and persisted to a SpeckleStream; subsequently, (2) they are retrieved in Grasshopper, where an
evolutionary optimisation algorithm devised is being run to search for the best possible building programme association. Once a satisfactory solution is found, it is being stored in SpeckleStream and subsequently retrieved in Dynamo and used to generate the associated layouts in Revit (Figure 35).

Another example coming from the industry usage of Speckle comes from Aurecon, a large general contractor. In order to showcase the company’s forward looking digital ambitions, they staged a fully integrated design-to-manufacture workflow of a sculpture wrapping around an existing column. In this specific example, Speckle played a central role in the data transactions involved in the process.

Figure 35: Shows the process behind Dialog’s building programming optimisation workflow. Image rights: Mark Cichy & Dialog (2018).
Figure 36: Still from Simon Yorke’s video describing the workflow enabled by Speckle. Image rights: Simon Yorke & Aurecon (2018).

The full workflow consisted of using 3d scanning to capture in as much detail as possible the site of the intervention. This information was used thereafter as the basis for a parametric model of the designed object. This parametric model was articulated in such a way that it allowed for aesthetic variation, while at the same time constraining the geometry within the fabrication parameters (specifically, 3d printing for joints and laser cutting for the wooden panels). The model was evaluated visually in virtual reality through a custom developed unity application. It was simultaneously evaluated for manufacturability and structural rigidity by different applications running parallel.

Throughout the process described above, Speckle was used to propagate the required information for the various subtasks of the workflow (Figure 36). Specifically, from the parametric model, two several separately curated data sets were transmitted: one for structural analysis, and one for visualisation within the virtual reality environment. From the structural analysis software, a different data set—containing just the deformation results—was used to inform design decisions. Lastly, two different other streams were used to curate the manufacturing of the joints between the wooden plates, and the wooden plates themselves. These were retrieved on two different corresponding computers that were controlling the 3d printer and the laser cutter.

Summarising, the workflow described above leveraged the potential to efficiently join multiple processes through lean, curated data sets. If this workflow were to rely on files, the overall speed and efficiency would have been compromised, as any
design change would have implied a time consuming re-export and re-evaluation. By curating for each sub-process just the data that it requires to perform its task, Simon Yorke, and Aurecon, were able to assemble a real-time feedback loop between design, analysis, and, as an ultimate step, manufacture.

5.6.2 Staging Workflows

Another case study was staged at the SimAud 2018 conference, as part of a day long workshop given by the author together with Paul Poinet, a fellow InnoChain researcher, at the TU Delft Faculty of Architecture. The workshop’s goal for the first half of the day was to enact a digital design workflow between several stakeholder roles in a design office: planning, architecture, analysis and reporting. This was achieved through two exercises, a guided one in which each individual team’s role and model was defined in advance, and an unguided one. The former was based on the multi scalar modelling process of an entry for the Tallinn Architecture Biennale pavilion, originally designed by Paul Poinet and Tom Svilans. Participants were split into teams, each controlling various aspects of the design (Figure 37): (1) master surface, (2) branching network, (3) network optimisation, (4) glulam blanks generation and (5) lamella generation.

![Figure 37: TU Delft SimAud 2018 guided exercise workflow map (above) and actual resulting model (below).](image)

While the guided exercise aimed to familiarise the participants with the basics of a multi-model approach to design data exchange, the unguided example did not provide a predefined organisational structure of the workflow. It proposed an exercise that entailed the conceptual design of three airport structures, namely two terminals (T1, T2) and a central mobility hub (T3) interacting with both; the
surroundings model was provided beforehand via an initial SpeckleStream. While initially split into three teams, communicative exchanges evolved continuously throughout the limited time dedicated to this exercise (Figure 38). For example, in the case of T1 needing to coordinate with a specific design element coming from T3, a person working on T1 would ask “can you send me the access points to the underground level of the station?”, moment in which the person from T3 would select from his model the specific design data relevant and create a new SpeckleStream specifically for the person from T1. Within the teams, similar exchanges ensued as the participants divided tasks amongst themselves, such as “can you send me the exterior envelope so I can do the façade panelisation?” or “I need the slabs of T2 to make sure the levels match with T1”, etc.

Other communicative exchanges were enabled through the composition of ad-hoc defined queries and projections on the SpeckleStreams being shared. The SpeckleServer supports filtering a collection of objects based on their properties and sub properties in order to allow for retrieving new sub-classifications that are user defined beyond the initial grouping embodied by the SpeckleStream. Essentially, a query can be seen as a request to the server to return only the objects matching certain criteria; for example: “from stream X, please return only the objects of type `Curve` with an `area` property larger than 120”, or “from stream X, please return only the

Figure 38: Diagrammatic representation of the evolution of the communication flows during the unguided exercise of the workshop.
objects with a `level` property equal to `+0.5_lower_basement``. Furthermore, a projection allows for the user selection of the fields returned. For example, if one actor would be interested in just general costing, he would request that only the “area” and “volume” fields of the objects would be returned, if present. These mechanisms were used to produce some simple quantity reports during the workshop by importing a set of such queries and projections in Excel, whereby total project areas could be centralised and costed.

### 5.7 Observed Classification Patterns & Performance

Mining the data from the database behind the experimental public test SpeckleServer, *Hestia*, further informs the nature of the emerging object-based classification patterns that the object-based methodology described in this chapter allows. The collected data allowed us to derive statistics on the amount of subdivision in a given file, or model. Throughout this section, the terms model and file are used interchangeably.

#### 5.7.1 Model Subdivision & Reassembly

The chart below shows the distribution frequency of sources and receivers per unique model (Figure 39). In order to cross-reference the data needed for this analysis, the database was queried for the number of *SpeckleStreams* originating from each model (black), for the number of *SpeckleStreams* being received in that model (red), and the combined number of sources and receivers (red). Finally, this dataset was partitioned into finite amount of frequency bins, spanning sequentially from one to twenty (i.e., models with one source, two, three, ..., twenty; same for models with one received stream, etc.) and models with more than twenty sources (and, separately, received streams).

An important caveat is that the data underpinning the following observations is limited by the fact that the Speckle plugins, in order to track these distributions, rely on the host application reported document UUID (or equivalent). This is not fully accurate, as some application APIs do not expose a method for retrieving it and, as a consequence, the plugins fall back on the file’s name. Furthermore, in the case that the file is not yet saved (it is newly created), a default value of “unnamed” is used as a document UUID, and such instances were removed from the data used in the analysis.
This chart shows a strong presence of models with only one SpeckleStream being sent, nevertheless with the majority of models originating in the range of two to ten different SpeckleStreams. Similarly, the majority of models receive two to ten different SpeckleStreams. The combined metric (sources and receivers per model) shows a similar distribution to the previous two.

This specific dataset is revealing a cohesive picture of how models are subdivided into different classifications and then subsequently regrouped into other files. The dominant one source/receiver per stream frequency score can be interpreted in three different ways. First, it can be seen as a mirror of the current de-facto approach employing monolithic files. Second, as a signal that approximatively one third of the design files do not need to be subdivided into more than one classification; in other words, they are already representing a productive categorisation that is self-sufficient. Third, that the model contained relevant data to other stakeholders that warranted only one classification—the rest being locally relevant information from either a technical point of view (in other words, software-specific “scaffolding”, geometric modelling helpers, etc.) or domain specialised data with internal meaning only.

<table>
<thead>
<tr>
<th></th>
<th>Streams sent per model</th>
<th>Streams received per model</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>2.47</td>
<td>2.29</td>
<td>2.78</td>
</tr>
</tbody>
</table>

Figure 39: Distribution of senders (black), receivers (blue), and both (red) per model. Vertical axis: number of models (logarithmic scale).

Figure 40: Distribution of sources per model.
Overall, the mean value of senders per file is 2.47, while the average number of receiving streams per model is 2.29 (Figure 40). This shows that on average, re-assembling different data sources is almost as prevalent within the design process as sub-classifying it into different logical groupings. Lastly, combined, the average number of data sources per model, irrespective of whether they are senders or receivers, is 2.78. In the case of a file-based approach, as currently is the accepted approach to design data communication, conceptually this measure would be one, except in the case of federated coordination models, in which case it would amount to the number of disciplines contributing to it (e.g., structures, mechanical, electrical, architectural, etc.). Nevertheless, even if this is a “rough” measure, it validates empirically the need to break down monolithic files into various sub-classifications based on the needs and requirements of the end-users’ task at hand: it shows that end users curate their own “federated model” on an as-needed basis if they are given a tool to do so.

5.7.2 Correlating Sources and Receivers

Overall, there are 11,495 total sources (each corresponding to one individual stream\(^\text{32}\)), and 26,019 total receivers, embedded within five different host applications used in the design process. In this analysis, only data from experimental or proprietary clients developed by other parties (such as Excel and ETABS) has been culled as their correctness of implementation cannot be verified. Figure 41 breaks down by application the number of sources, or streams created from, and receivers, or number of streams received into that specific application. The last column, Ratio R/S, shows whether the respective application has been used more for sending data, or for receiving data, by calculating how many receivers there are for one given sender.

\(^{32}\) In total, there are 28,262 streams present in the database. The discrepancy between the number of streams and sources is explained by their (1) versioning and history, and (2) their direct programmatic creation (in which case sources and receivers do not get registered by the server).
<table>
<thead>
<tr>
<th>Source</th>
<th>Sources (S)</th>
<th>Receivers (R)</th>
<th>Ratio R/S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rhino</td>
<td>866</td>
<td>725</td>
<td>0.84</td>
</tr>
<tr>
<td>Grasshopper</td>
<td>4,919</td>
<td>9,611</td>
<td>1.95</td>
</tr>
<tr>
<td>Dynamo</td>
<td>2,707</td>
<td>12,801</td>
<td>4.73</td>
</tr>
<tr>
<td>GSA</td>
<td>2,445</td>
<td>2,882</td>
<td>1.18</td>
</tr>
<tr>
<td>Revit</td>
<td>558</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Total</td>
<td>11,495</td>
<td>26,019</td>
<td>2.26</td>
</tr>
</tbody>
</table>

*Figure 41: Breakdown of sources & receivers by application type.*

Individually, one can see that the two visual programming environments present, Grasshopper and Dynamo, have been used more for receiving (or recoupling data) than for sending data (with a R/S ratio of 1.95 and 4.73, respectively). In the case of Dynamo, the high R/S ratio confirms the observed workflows in the industry, whereby it is used to pull in data from other software, process it, and thereafter embed it into Revit. While Rhino comes across as primarily a source of information (with a R/S ratio of 0.84), GSA, a structural analysis software, has a balanced ratio (value 1.18), tipped towards receiving: this implies that data from multiple sources is used to assemble a structural analysis model.

Notably, the most important finding is the average R/S ratio across all applications, namely 2.26. Some sources potentially have no corresponding receiver, or, in other words, are not consumed by any party (albeit they can be used just for visualisation purposes online, in which case this is not tracked by the server); nevertheless, the table above includes this scenario. As such, the 2.26 receiver-to-sender ratio implies that there are, on average, for every given sender, two or more receivers. Lastly, by linking with the case made for data curation earlier on in this chapter (*Section 5.2, From Sharing Files to Curating Data*), this ratio entails that, within the limitations of Speckle, information is being shared and consumed in a productive manner by end-users and data is, on average, not being shared without intent and, implicitly, meaning.

### 5.7.3 Storage Performance

This analysis compares the performance from the point of view of storage space needed of a file-based approach and an object-based approach to design data classification. In the first table, presented below, a 3D Rhino model of the Villa Savoye
is used as an experimental base (Figure 42). The model is split into the following categories, similar to examples seen in practice: (1) all, (2) structure, (3) walls, (4) auxiliary (furniture, windows, doors), (5) ground floor, (6) top floor, and (7) roof. Simultaneously, analogous SpeckleStreams using the same classification are created. These categories, while logically distinct, are not mutually exclusive (e.g., the ground floor classification (5) contains objects from (2) structure, (3) walls, and (4) auxiliary).

<table>
<thead>
<tr>
<th>Villa Savoye</th>
<th>Accumulated Size</th>
<th>% of total</th>
<th>Accumulated Speckle DB Size</th>
<th>% of total</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) all</td>
<td>39,067,396</td>
<td>100%</td>
<td>36,017,005</td>
<td>100%</td>
</tr>
<tr>
<td>(2) structure</td>
<td>43,027,966</td>
<td>110.14%</td>
<td>36,037,020</td>
<td>100.06%</td>
</tr>
<tr>
<td>(3) walls</td>
<td>53,075,937</td>
<td>135.86%</td>
<td>36,062,421</td>
<td>100.13%</td>
</tr>
<tr>
<td>(4) auxiliary</td>
<td>59,114,033</td>
<td>151.31%</td>
<td>36,092,763</td>
<td>100.21%</td>
</tr>
<tr>
<td>(5) ground floor</td>
<td>67,470,069</td>
<td>172.70%</td>
<td>36,130,856</td>
<td>100.32%</td>
</tr>
<tr>
<td>(6) first floor</td>
<td>77,698,734</td>
<td>198.88%</td>
<td>36,179,276</td>
<td>100.45%</td>
</tr>
<tr>
<td>(7) roof</td>
<td>82,339,127</td>
<td>210.76%</td>
<td>36,231,523</td>
<td>100.60%</td>
</tr>
</tbody>
</table>

Figure 42: Total storage space performance in a multiple sub-classification scenario on an example model.

The “Accumulated Size” column shows, in bytes, the total storage occupied by the individual sub-classifications of the model as they are progressively created. The “Accumulated Speckle DB Size” shows the total storage size of the database as the individual SpeckleStreams (containing the same objects as their equivalent file-based sub-classification) are being persisted. Simultaneously, the normalised value of the accumulated sizes is calculated as a percentage of the total corresponding to the two individual methods used in order to remove the bias introduced by the two different serialisation methods.

Initially, both the file-based approach and Speckle start with a similar value in terms of size (~39mb and 36mb, respectively). Nevertheless, as the model is sequentially split into multiple categories, one can observe a large deviation between the two methods: when all seven sub-classifications are in place, the file-based approach
utilises ~82mb, whereas Speckle still only slightly more than ~36mb, essentially amounting to storage savings of almost 50% (~86mb vs ~36mb). This striking difference results from the fact that Speckle, when storing a new classification, first checks whether the objects that make it up already exist in the database (based on their hash, as discussed in Section 4.1 and 4.5), and if they do, does not store them again, only referencing the pre-existing object in the newly created SpeckleStream.

To corroborate the results from the previous table, the same methodology is applied on a Rhino file provided by an industry partner with a similar number of sub-classifications (Figure 43), specifically (1) the whole model, (2) structural elements, (3) façade, (4) raster (grid), and (5) interior (all project and office specific naming conventions have been removed in order to anonymise the source file).

<table>
<thead>
<tr>
<th>Anonymous Model</th>
<th>Accumulated Size</th>
<th>% of total</th>
<th>Accumulated Speckle DB Size</th>
<th>% of total</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) all</td>
<td>27,394,489</td>
<td>100%</td>
<td>38,372,903</td>
<td>100%</td>
</tr>
<tr>
<td>(2) structure</td>
<td>33,392,349</td>
<td>121.89%</td>
<td>38,382,505</td>
<td>100.03%</td>
</tr>
<tr>
<td>(3) &quot;fassade&quot;</td>
<td>50,523,295</td>
<td>184.43%</td>
<td>38,392,510</td>
<td>100.05%</td>
</tr>
<tr>
<td>(4) raster</td>
<td>51,396,360</td>
<td>187.62%</td>
<td>38,396,094</td>
<td>100.06%</td>
</tr>
<tr>
<td>(5) interior</td>
<td>57,204,093</td>
<td>208.82%</td>
<td>38,411,053</td>
<td>100.10%</td>
</tr>
</tbody>
</table>

Figure 43: Total storage space performance in a multiple sub-classification scenario on a model coming from a real life project.

This specific model has a different starting point: initially, the file size is smaller than its corresponding equivalent from the Speckle server’s database (~27mb and ~38mb, respectively). Nevertheless, as more sub-classifications are added, the same overall size savings trend in favour of the object-based approach can be observed as in the previous table (Figure 42).

Summing up, the cost of sub-dividing a model in a file-based approach is approximatively twice as high compared to an object-centred approach, as implemented in Speckle, in the scenario where the sub-divisions are not mutually exclusive. This last assumption is a limitation of this assessment: while, from a common sense point of view, it stands true, further work would be needed to understand the actual overlap between classifications, and, subsequently, storage
performance of the two approaches assessed. Nevertheless, persisting a new classification on an existing base adds less than 0.1% to the overall storage size, showing that, with an object-based classification methodology, the actual cost of creating new logical groupings of objects is negligible.

5.8 Conclusion

This chapter investigated how the way digital design data is stored and classified influences the communication process. The three layers of the Speckle framework—the presentation layer (various application plugins), the application layer (encoding routines and the server) and the storage layer (MongoDB database and its data model), combine together to orchestrate a different technology for design data communication than that offered by contemporary file-based approaches. These two technologies must not be seen as excluding one another: on the contrary, they coexist and complement each other, as the approach outlined here relies on design models and the software applications that produce them; to a certain extent it is an added technological layer that aims to support natural communicative processes.

The differences between the two, when observed in action, lead to separate communicative processes that have been the subject of the analysis underpinning the non-technical sections of this chapter. The findings can be summarised as follows:

- The communicative productivity of an object-based curatorial approach to design data classification has been validated through both qualitative means, by assessing several case studies observed “in the wild” (Section 5.6.1), as well as through empirical observations based on the observed usage of Speckle:
  - On average, design models were broken down into 2.47 separate sub-classifications, or SpeckleStreams (Figure 40). Furthermore, when taking into account sources coming into a given model, the average count of both sources and receivers per model was 2.78, highlighting a dynamic process of fragmentation and re-assembly (Section 5.7.1).
  - Speckle client usage data across five different applications showed that, on average, for each single source there are at least two receivers (specifically, 2.26) (Figure 41). This further confirms that curated information is being produced and consumed in an efficient manner (Section 5.7.2).

- An object-based approach to data persistence is potentially twice as efficient as a file-based one in enabling multiple overlapping classifications of design information (Section 5.7.3). Furthermore, the actual cost of creating new classifications on an existing base is virtually negligible (< 0.1% of the original
model size), which facilitates the emergence and evolution of efficient “informal” communication exchanges, as described in Section 5.6.2.

So far, the various transactional costs that Speckle implies have been discussed tangentially, such as the speed at which data is transmitted and stored, as well as the overall size of the information being exchanged. These matters are discussed in the following chapter, which assesses how to provide for, and optimise, the transactional requirements of the communication process in the context of digital design.
6. Data Transaction
or, Digitally Enabling Communicative Contracts
6.1 Nextness & Sequentiality

The previous two chapters challenged the status quo of data representation (how information is structured) and storage (what information is transmitted, and how it is classified) within the AEC industry. This research asserts that an object-centred design data sharing model can support organic communicative topologies to emerge, thus befitting natural dialogue. Nevertheless, simply storing and retrieving information is not enough as actors rely on extra-informational mechanisms to convey signals regarding the status of the communication itself, for example, whether the message has been successfully received or interpreted, whether that information was further used for other purposes, whether it caused confusion or clarity, etc. Consequently, this chapter looks at what is the best way to enable efficient and meaningful design data transactions from a technical point of view. Furthermore, this analysis is coupled with a qualitative assessment of end-user requirements in terms of interaction (and interfaces) needed to support their workflow and how these are reflected, limited or enhanced by the digital infrastructure available.

Social communication processes—human dialogue—have a set of requirements that enable their correct functioning. Beyond the most evident one, that there need to be two or more participants for communication to happen, there is nextness. As discussed earlier on in Chapter 2, nextness, or immediacy, is a key ingredient of a productive communication process. Essentially, it entails that messages between actors should not spread out in time between themselves as, by staggering them too far apart, their relevancy is compromised. A hypothetical example is person A asking B “What time does the train to Loughborough depart?”, and B providing the answer after the train has already left, or, generalising, after a delay that negates the relevancy of the information she or he provides to A.

For example, a long pause before an answer to a question can be interpreted by the sender as either reservation, confusion or doubt as to relevancy of the communication on behalf of the receiver. These mechanisms, while originally derived from the observation of face-to-face dialogue and interaction, have nevertheless analogues in digital communication systems, from chat applications to code collaboration.
platforms. For example, WhatsApp, Facebook Messenger, and many other similar offerings convey to their users, through various interface mechanisms, whether a certain message has been successfully sent, whether it has arrived at the recipient, and whether it has been actually read. The capacity of such systems to convey meta-information enables a much richer, satisfying and efficient digital dialogue to happen. For example, if person A sends an urgent inquiry to person B, and, after one hour of waiting, A can see that B still didn’t read the message, A can now decide to make a phone call in order to resolve the situation. Github, a collaborative platform for writing software, has similar mechanisms for notifying users of various informational transactions, such as code modifications, issues, reviews, comments, pull requests, etc.

Institutional or professional communication, of course, does not necessarily have the same strict temporal requirements as informal, person-to-person dialogue. Within such a context, actors work in parallel over spans of time that can be long or short depending on the requirements of their common activity, and synchronise (or coordinate) efforts using a different rhythm than direct interpersonal exchanges. However, in this scenario, a concept parallel to nextness takes precedence: sequentiality. As mentioned previously in Chapter 2, Garfinkel, posits that an act of communication is an ordered sequence of information exchanges (Garfinkel and Rawls, 2006) that, in time, create a shared context for the communicants. This allows them to situate codified information and consequently, as Sperber and Wilson note, optimise the relevancy of their exchanges (Wilson and Sperber, 2008) as well as constructing a sense of shared understanding (Mondada, 2011).

The examples above are meant to briefly underline the capacity of the digital medium to shape and enforce communicative contracts by making the sequentiality of information exchanges between actors visible and, to a certain extent, tangible. Referring back to the first example wherein A asks B at what time the train to Loughborough departs, and B, ignoring A for a period of time, replies after the train has already left, thus rendering the exchange useless, one can state that B, by deferring his answer, violated the communicative contract commonly expected of natural, interpersonal dialogue by deliberately breaking its sequence. In face-to-face interactions, these transgressions (or compliances) are self-evident or implicitly facilitated by the reality in which the conversation is happening and the social norms by which the participants abide. Nevertheless, in the case of institutional dialogue (i.e., two or more organisations collaborating), communication contracts need to be explicitly defined as protocols. In the case of the digitally enabled information exchanges, these protocols need to be technologically fulfilled by providing instruments for making the order and relationship between information exchanges
tangible.

Current collaboration workflows, as specified by BIM methodologies, do not lend themselves easily to design data informed dialogue. The problem is two-fold, but both its aspects share a common root. First, the speed of digital data transactions is limited by the size of the discipline models, or files, that are used, fact which is acerbated when taking into account potential limited internet connectivity speeds. Consequently, these design files and models, because of their size, impose limitations on the immediacy of the transactions involved thus leading to a well demarcated, formal process that does not lend itself easily to the parallelisation of work. Coordination, seemingly, happens once a week by popular consensus (van Berlo et al., 2012). Second, file-based transactions (or model based) encode a high number of changes in bulk in a non-transparent manner. As follows, it is difficult to extract the sequences, or causal relationships, between the exchanged information as it is not explicitly available. For example, an architectural design model, over a week, accumulates a number of changes in the layout of the building. Some of these are the direct consequence of changes and insights gathered from the structural model. Simultaneously, during the same week, the structural model undergoes various other changes coming from the MEP model. As follows, these changes, because they are not immediately available to all interested participants, have the potential to negate each other: the architect worked on stale data, thus creating a layout that does not respect the structural model due to its simultaneous modification resulting from the installation of a new water column in the MEP model. Furthermore, because there is no clear relationship diagram available between the models themselves, the participants cannot judge the impact of their changes and cannot determine whether certain parts of the overall design need to be updated to reflect the latest information.

To better understand the dynamics of multi-disciplinary teams and validate the claims above, as part of the living laboratory context within which this research project was undertaken, a workshop on practice-based workflows was facilitated. Therein, several key engineers and project managers from Buro Happold (engineering consultancy) and Rhomberg Sersea Group (construction developers) were invited to describe the usual process that a project would go through in terms of data exchange pertaining to certain design tasks that they were familiar with. The participants were guided through a set of storytelling exercises in which they would describe specific projects they were involved with, but as well “roleplay” other actors from within the teams with which they were involved.
Figure 44: Sample storyboard showing conception stages in the engineering process, from early stage design (sketches, hand calculations and simple analysis models) through the middle stages, where the BIM model is actually composed, and finally to final, detailed design stages.

Figure 45: Left: Architect-Engineer workflow during the design and analysis of a complex interior staircase project. Right: Workflow for a bid costing process undertaken by a developer and its subsidiaries. In red, is the measurement based process (quantities), and in blue the activity based process (site planning & construction management).

The workshop yielded, in total, ten user stories involving roles such as planner, quantity surveyor, design coordinator, engineer, as well as field operatives. These user stories, or workflow maps, spanned from finite, contained design tasks such as the structural optimisation and specification of a complex stairway to project costing and bid creation, as well as on-site project delivery assessment. For example, a planner wants to associate planning activities to design objects to be able to plan activities in planning software (e.g., Tilos, Primavera P6, Bentley Synchro, Microsoft
Project, etc.); an estimator needs to create a model view on the design scope in order to bring it into an estimation software and create a cost estimate; a design coordinator collects and federates the latest BIM models to assign design tasks/issues back to the individual consultants (Figure 45); a structural engineer needs to have different view of the same data so in order to concentrate only what is important at any given time (Figure 44).

Each discipline, and even each actor, navigates within their own ecosystem of tools that act as an extension to their knowledge and skills. The participants highlighted the fact that manual interventions are required throughout the process. Two time-consuming issues were highlighted. The first is data conversions between the various proprietary formats and subsequent cleaning and pruning of information. This can partly be attributed to the segregated and siloed nature of the different software packages that they are using to achieve their tasks. This results in slow design iteration cycles due to the large amount of information being transferred, its lack of usual suitability for the task at hand as well as its incompatible formats and potential errors. The second issue raised was extensive coordination with other team members to ensure data consistency across all the actors: does the seismic analysis model incorporate the latest changes from the architect’s model? did the architect incorporate the changes from the lateral loads analysis? does the architect incorporate the latest specified construction method? was the project schedule updated with information from the latest coordination model? etc. Common in many areas, this process is seen as “data wrangling”, or “data massage”. Ultimately, it causes slow update cycles—resulting from the friction of data conversion and processing—that, in turn, lead to a weakening of nextness: instead of being close together, communicative exchanges are far spread apart and, as such, lose their relevance in the minds of the participants.

Another cause is the lack of a tangible sense of sequentiality in a digital communicative environment. In a normal, face-to-face situation, participants to a conversation would be able to follow along the issues being discussed by virtue of the order in which the they speak: each speaker’s utterances follow, and depend on (or react to) the previous ones. In a digital conversation, that may span multiple geographic areas and their associated time zones, one cannot rely on the same heuristic, or embedded mechanisms to understand the sequence of informational exchanges as present in the real world. Firstly, this is because participants are not in the same physical space. Secondly, because, unlike a human conversation, digital conversations happen quasi-simultaneously and overlap in various layers: while participating in a larger data exchange environment pertaining to a wider aspect of the design problem, one can have in parallel smaller, more directed conversations pertaining to various sub-issues with different stakeholders. Finally, sequentiality is
befuddled because the file-based approaches currently employed cause the
ingformation to be detached from its source: once an export is made, it has no links
back to the model that originated it; it becomes “disembodied”.

As follows from the analysis above, two technical research directions become evident.
The first tackles the issue behind the lack of nextness in digital design
communication, and specifically looks at whether one can decrease the spacing
between data exchanges by minimising their size by decoupling them from the
magnitude of the model (or file) itself. Second, this chapter investigates how the
sequentiality of digital information exchanges between actors can be expressed and
revealed to the participants in order to reduce the intensive coordination
requirements resulting from detaching the transmitted information from its source.

6.2 Optimising Transactions: Differential Updates

6.2.1 Differential Updates

The previous chapter described the three main sections of the system architecture of
the research instrumentation, namely the presentation, application and persistence
layers and provided a general overview of the way they interact. Now, this section
addresses the way design data transactions between these layers is implemented.
While there are many points of interaction that facilitate various informational
exchanges between the systems, the most important (by volume) aspects relate to the
transfer of design information (geometry—points, lines, etc.—and associated
metadata) between a client application that generates the data, the database and
another client application that consumes the data.

Document- or model-based approaches to communication propagate changes in
bulk. That is to say, if one object of the model is modified, then the whole model is
re-sent to the persistence layer in order to propagate one single change.
Consequently, this leads to sub-optimal transactions: even if the actual modification
is small, the amount of data being transferred directly depends on the size of the
model. This results in waiting times that are longer than necessary, even in the
context of a collaboration system deployed on a local network (for example, inside a
company’s offices). In the case of a distributed system, i.e., where the system relies
on digital networks infrastructure, this problem is even more present due to
increased latencies and middle-ware handling of the routing and distribution of
information packets across a geographically dispersed network. Moreover, longer
transaction times limit a communicative act by impeding its *immediacy* due to the
fact that, in the end, the speed of the transactions is directly proportional to the size
of the model, and not of the change effected.
Speckle, being object-centred, allows us to implement a differential compression layer within the transaction logic of the API clients (at both the sending and receiving end). In essence, this allows us to identify and transmit only the changes between the various states of the object collection (SpeckleStream), thus reducing the transmitted information to the minimum required. This is possible by leveraging a local cache of objects that have been already sent or received. Initially, this cache was implemented “in-memory”, in other words it does not persist after the client application that authored or received the objects has been closed. In later, more mature implementations, this cache resides locally on the user’s hard disk in a SQLite database that enables checking against both outgoing data (objects being sent) as well as incoming data (objects being received), across multiple authoring applications, and across an unbounded amount of time, not restricted to computer or host application restarts.

For example, when creating a new SpeckleStream \( x \) consisting of entirely new objects \((A, B\) and \(C)\), the original state of \( x \) is “empty” \((x = \emptyset)\). Consequently, all objects will be sent to the SpeckleServer in order for them to be saved in the database, and thereafter \( x = \{A, B, C\} \). Through a subsequent modelling operation, objects \(B\) and \(C\) are modified and, because of the immutable approach to object identity based on unique hashes of their properties described in the previous chapter, they become \(B^*\) and \(C^*\); \(A\) nevertheless stays unchanged. Next, by performing a set intersection between the previous state of \(x\) \((\{A, B, C\})\) and its new local state \((\{A, B^*, C^*\})\) one can assemble the actual content of the update transaction as being \(U = \{\text{ref } A, B^*, C^*\}\). \text{ref } A is understood as a purely symbolic reference to the original object, and not its actual content; this approach is needed in order to maintain the correct order of the objects inside the collection and is a simpler and leaner implementation (for some cases) as opposed to providing extra information with the update payload as to what objects to remove or insert and from which index. At the end of the process, the SpeckleStream \( x = \{A, B^*, C^*\} \). In the meantime, each successful update transaction has also been recorded in the local cache. After the stream creation, the cache contained only \(\{A, B, C\}\), whereas by the end of the second update, it increased to \(\{A, B, C, B^*, C^*\}\).

The cache can be seen as a ledger in which one can keep track of which objects already exist in the persistence layer. Consequently, it can be leveraged across multiple update operations coming from different locations in order to optimise the transaction size. For example, if one were to create a new SpeckleStream, \( y \), containing objects \(B^*, B\) and \(C\), because the system knows that these objects already exist in the persistence layer, the update payload will only contain a list of symbolic references to \(B^*, B\) and \(C\), without the actual object content, thus greatly speeding up
the transaction speed. Furthermore, an equivalent data differencing approach can be applied when receiving information, or updating a SpeckleStream. For example, a client receives information from SpeckleStream $x$. Initially, this contains three objects, namely $\{A, B, C\}$ which are stored upon retrieval in the local cache. Subsequent updates from the authoring application add several objects to $x$, so that its status is now $\{A, B, C, D, E, F\}$. When updating $x$, the client will cross-check with the local cache to see if an object already exists before requesting it from the server and potentially incurring a high transactional cost. In the current scenario, because the cache already contains $\{A, B, C\}$ the only objects that are actually transferred between the persistence layer and the receiving client are $\{D, E, F\}$. Similarly to the updating process, this results in a large reduction of the overall size of information being transferred, thus increasing the speed with which an application that consumes data can enact an update.

Simply put, this approach can be described as a client-side differential compression algorithm operating at the level of design data object sets (SpeckleStreams). It leverages the approach to object identity and immutability described in the previous chapter to incrementally create a shared informational context which is thereafter used to reduce information redundancy and consequently the amount of information being transferred. Similarly to natural dialogue, where participants rely on their shared context in order to increase the relevancy of their exchanges, this approach leverages a technically created context for speeding up design data transactions.

**6.2.2 Observed Performance**

So as to be able to quantify the impact on data transactions that differential updates have, this section presents an analysis of the transactions associated with the streams from a live project in which Speckle was used to coordinate the design process between several disciplinary groups. The full process and the relationship between the stakeholders involved is described in more detail, as well as qualitatively assessed, in Section 6.5.3, Design and Façade Engineering.

The first table compares the total and average size of the first part of a transaction, namely the one in which the receiver sends the data, in two distinct scenarios (Figure 46). The first, namely the “Delta” scenario, describes the actual way Speckle works, and leverages the differential update mechanism described in the previous section. The second, the “Bulk” scenario, discounts this and assumes a file-based approach in which the current state of data is always shared in full.

The transaction size in both scenarios was calculated based on the average object size as reported by the database, namely 4.5 kb (see Section 4.3). It is important to note
that this eliminates the bias introduced by employing different serialisation methods (i.e., pertaining to the data format used). The time column shows, hypothetically, how much an average upload would take based on UK national averages on internet speed (specifically, an upload speed of 7 Mbps, or 875kb/s)\(^{33}\). Lastly, the final column shows the rapport between the average time of a Bulk scenario transaction versus a Delta scenario one. Overall, across the eight streams that were analysed, the total number of recorded transactions is 122, spread across a period of approximately two weeks\(^{34}\).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>stream A</td>
<td>8,586.90</td>
<td>505.11</td>
<td>0.58</td>
<td>18,701.30</td>
<td>1,100.08</td>
<td>1.26</td>
<td>217.79%</td>
</tr>
<tr>
<td>stream B</td>
<td>902.40</td>
<td>150.40</td>
<td>0.17</td>
<td>2,284.20</td>
<td>380.70</td>
<td>0.44</td>
<td>253.13%</td>
</tr>
<tr>
<td>stream C</td>
<td>2,472.20</td>
<td>274.69</td>
<td>0.31</td>
<td>3,064.40</td>
<td>340.49</td>
<td>0.39</td>
<td>123.95%</td>
</tr>
<tr>
<td>stream D</td>
<td>67,087.80</td>
<td>13,417.56</td>
<td>13.33</td>
<td>193,809.20</td>
<td>38,761.84</td>
<td>44.30</td>
<td>288.89%</td>
</tr>
<tr>
<td>stream E</td>
<td>876,625.20</td>
<td>16,233.80</td>
<td>18.55</td>
<td>1,130,411.10</td>
<td>20,933.54</td>
<td>23.92</td>
<td>128.95%</td>
</tr>
<tr>
<td>stream F</td>
<td>45,369.10</td>
<td>5,671.14</td>
<td>6.48</td>
<td>108,993.00</td>
<td>13,624.13</td>
<td>15.57</td>
<td>240.24%</td>
</tr>
<tr>
<td>stream G</td>
<td>37,722.20</td>
<td>6,287.03</td>
<td>7.19</td>
<td>69,146.40</td>
<td>11,524.40</td>
<td>13.17</td>
<td>183.30%</td>
</tr>
<tr>
<td>stream H</td>
<td>36,711.70</td>
<td>4,079.08</td>
<td>4.66</td>
<td>75,435.00</td>
<td>8,381.67</td>
<td>9.58</td>
<td>205.48%</td>
</tr>
<tr>
<td>average</td>
<td>134,434.69</td>
<td>5,318.84</td>
<td>6.08</td>
<td>216,063.70</td>
<td>11,880.09</td>
<td>13.58</td>
<td>205.22%</td>
</tr>
</tbody>
</table>

Figure 46: Comparison table between transactions in a differential update approach (Delta) and a bulk sharing approach.

The table shows that the overall Bulk approach ranges between being 123% (stream C) and 288% (stream D) larger than the Delta approach, averaging at 205%. Streams C and E exhibit a lower Bulk to Delta ratio, signifying that the changes that these sub-classifications underwent were major. However, without employing differential updates, the average cost of the first half of a transaction, namely sending, is twice as high as with. This cost is incurred solely by the sender.

The second part of a transaction, namely receiving, has a different cost and is incurred


\(^{34}\) Only the transactions that were explicitly saved by a user (as a stream version) could be analysed. Furthermore, transactions that do not show any change have been excluded.
multiple times by each stakeholder that receives the data being transmitted. As shown in Section 5.7.2, Correlating Sources and Receivers, on average there are 2.26 receiving parties for each sender. Nevertheless, proportion wise, this does not affect the observed efficiency of the differential update method as compared to the bulk one; it only influences the absolute amount (size) of data being transferred.

In order to highlight the dynamic nature of data transactions, the chart below presents a view on the 15 recorded transactions from Stream A in chronological order (Figure 47). Each transaction is broken down by number of objects added (green), removed (red), and common (grey).

![Figure 47: Transaction breakdown: green represents newly added objects, red removed, and grey unchanged.](image)

The chart above shows a high range of dynamism, from small changes (transaction 11) to complete “re-writes” in which all, or almost all, the objects are removed and replaced with newer ones (e.g., transaction 6 and 12). In order to quantify the impact of this process, the table below shows an expanded view of each transaction (Figure 48). Specifically, the second, third and fourth columns (Added, Removed, and Common) detail the nature of the transaction broken down by the number of objects involved in the respective operation. The Delta Size column shows the estimated amount of data transferred based on an average object size of 4.7kb\(^{35}\) and the amount of “Added” objects, which are all assumed as being unique\(^{36}\). The Bulk Size column

---

35 This figure is different (but not by much) than the one presented in Chapter 4 as it is always in flux, depending on the objects present in the database.

36 This potentially favours the “Bulk”, file-based approach, as this is not always the case: for example, adding a previously removed set of objects.
is calculated based on the total number of objects per stream. Lastly, the *Bulk/Delta* column shows the ratio between the previous two.

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Added</th>
<th>Removed</th>
<th>Common</th>
<th>Total</th>
<th>Delta Size (kb)</th>
<th>Bulk Size (kb)</th>
<th>Bulk/Delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>215</td>
<td>0</td>
<td>0</td>
<td>215</td>
<td>1010.5</td>
<td>1010.5</td>
<td>100.00%</td>
</tr>
<tr>
<td>2</td>
<td>70</td>
<td>78</td>
<td>137</td>
<td>207</td>
<td>329</td>
<td>972.9</td>
<td>295.71%</td>
</tr>
<tr>
<td>3</td>
<td>65</td>
<td>65</td>
<td>142</td>
<td>207</td>
<td>305.5</td>
<td>972.9</td>
<td>318.46%</td>
</tr>
<tr>
<td>4</td>
<td>76</td>
<td>84</td>
<td>123</td>
<td>199</td>
<td>357.2</td>
<td>935.3</td>
<td>261.84%</td>
</tr>
<tr>
<td>5</td>
<td>134</td>
<td>136</td>
<td>63</td>
<td>197</td>
<td>629.8</td>
<td>925.9</td>
<td>147.01%</td>
</tr>
<tr>
<td>6</td>
<td>194</td>
<td>187</td>
<td>10</td>
<td>204</td>
<td>911.8</td>
<td>958.8</td>
<td>105.15%</td>
</tr>
<tr>
<td>7</td>
<td>64</td>
<td>77</td>
<td>127</td>
<td>191</td>
<td>300.8</td>
<td>897.7</td>
<td>298.44%</td>
</tr>
<tr>
<td>8</td>
<td>213</td>
<td>189</td>
<td>2</td>
<td>215</td>
<td>1001.1</td>
<td>1010.5</td>
<td>100.94%</td>
</tr>
<tr>
<td>9</td>
<td>204</td>
<td>205</td>
<td>10</td>
<td>214</td>
<td>958.8</td>
<td>1005.8</td>
<td>104.90%</td>
</tr>
<tr>
<td>10</td>
<td>57</td>
<td>57</td>
<td>157</td>
<td>214</td>
<td>267.9</td>
<td>1005.8</td>
<td>375.44%</td>
</tr>
<tr>
<td>11</td>
<td>84</td>
<td>22</td>
<td>192</td>
<td>276</td>
<td>394.8</td>
<td>1297.2</td>
<td>328.57%</td>
</tr>
<tr>
<td>12</td>
<td>266</td>
<td>269</td>
<td>7</td>
<td>273</td>
<td>1250.2</td>
<td>1283.1</td>
<td>102.63%</td>
</tr>
<tr>
<td>13</td>
<td>65</td>
<td>65</td>
<td>208</td>
<td>273</td>
<td>305.5</td>
<td>1283.1</td>
<td>420.00%</td>
</tr>
<tr>
<td>14</td>
<td>65</td>
<td>65</td>
<td>208</td>
<td>273</td>
<td>305.5</td>
<td>1283.1</td>
<td>420.00%</td>
</tr>
<tr>
<td>15</td>
<td>55</td>
<td>59</td>
<td>214</td>
<td>269</td>
<td>258.5</td>
<td>1264.3</td>
<td>489.09%</td>
</tr>
</tbody>
</table>

*Figure 48: Historical analysis of a stream’s transaction record.*

In the first transaction, 215 objects are added. Consequently, the two approaches yield a similar ratio. In this specific scenario, there is no advantage between the two approaches. Nevertheless, in the second transaction, 78 objects are replaced with a different 70, whilst 137 are the same. This leads to a much greater efficiency when employing differential updates: the bulk approach is almost three times bigger than its counterpart, as implemented in Speckle. Similarly to the former scenario, the 12th transaction holds only 7 common objects, with the bulk being replaced (205 removed, and 204 added). In this instance, the gain difference is minimal (the bulk approach is only 2% larger than the differential update one). Overall, the differential update method provides size (and, implicitly, duration) significant savings over the classic
approach, which may be 489% the size of an equivalent delta (Figure 48, Transaction 15). In other words, it can be five times faster and five times smaller.

6.2.3 Theoretical Best & Worst Cases

On average, differential updates allow for data transactions to be half the size as they would be if a classic, file-based methodology, was employed; it was observed empirically to be five times faster and leaner in size. To fully assess the performance of this method in its best- and worst-case scenario, the chart below shows the transaction history of stream E which includes exhaustively both (Figure 49). It is accompanied by a following table (Figure 50), showing two exemplary transactions, one “worst-case”, and one “best-case”, derived in the same way as the previous one from Section 6.2.1, and containing the same headings.

![Figure 49: Transaction history for Stream E. Green represents newly added objects, red removed, and grey unchanged.](image-url)
Stream E’s transaction history is comprised in approximatively equal parts of minor changes and complete “re-writes”. Speculatively put, the end-user seems to be searching for a suitable design, completely changing the information being sent (the transactions with zero common objects) and as well thereafter surgically fine-tuning it (the transactions with only one added and removed object). For example, transaction number 32 is one of surgical fine-tuning, whereas transaction 36 is a complete overhaul of the data contained in the stream.

The expanded table detailing these two transactions shows the extremes in comparative performance between the approach utilising differential updates and the bulk, file-based one. In the worst-case scenario, the two are equivalent, with a Bulk/Delta size ratio of 1 (100%). Nevertheless, transaction 36 shows a staggering improvement: because of the fact that the change size is small in comparison with the unchanged objects, the file-based approach is 158.700 larger than the differential update one. Mathematically speaking, the best-case scenario has no theoretical upper limit: if the change size is small enough compared to the whole, the transaction cost will be tending towards infinity.

Lastly, the file-based approach to data transactions can be seen as only having a “worst-case” by virtue of the fact that it always includes all the information, regardless of whether it was modified or not. In a social conversation, this would be akin to repeating all previous statements before adding one’s own. Nevertheless, the differential update method, as observed in its implementation in the Speckle plugins, decouples the transaction size from the model size, and allows for change-dependent digital design data transactions.

### 6.3 Assembling the Network (Instrumentation)

In order to provide a comprehensive overview of the communicative network and its topology as it emerges during the design process, an overview on where the information is coming from and where it is going towards is needed— in short, who are the senders and receivers. So far the term *client* has been used to denote the actors that interact with the *SpeckleServer* through the publicly exposed API, nevertheless there is no persistent trace of them. This means that if John updates stream X from a
design application using a specific client, one has no way of notifying Mary and Howard who consume design data from stream X that their information is now stale. Consequently, the technical instrumentation, Speckle, needs to incorporate a way of keeping track of (1) individual users and (2) the clients that they create in order to transact (send and receive) design data. Furthermore, it needs to enable a way in which information about the status of transactions can be pushed to the interested parties, for example “SpeckleStream X has been updated”, which will be discussed in the Propagating Information section.

6.3.1 Users

In order to identify which human actors create what design data, a basic user authentication system on the application layer is implemented, specifically within the SpeckleServer (as part of the API). In order to access and use the Speckle API, a person must register with the system. The details that need to be provided are kept to a minimum—namely email and password. There are other optional fields, such as full name and company. After registering with a specific instance of a running SpeckleServer, an end-user now has access to the full power of the API.

The implementation of authentication depends on an architectural choice between opting whether the SpeckleServer should be stateful or stateless. A stateful system implies sharing between client requests the user’s session details, whereas a stateless server must be able to perform authentication (or other operations) on a by-request basis. The former is easier to implement, nevertheless the system architecture is much more difficult to subsequently scale, which has an impact on performance in cases of high usage. Consequently, a stateless architecture is used, which essentially allows for an infinite number of SpeckleServers to be simultaneously run (nevertheless sharing one persistence layer) in order to meet the user demand that they encounter.

Consequently, this means that a user has to provide a proof that she or he is authenticated on every API call. A standard industry pattern is implemented by generating expiring JSON Web Tokens (JWT) upon a user login and the moment of initial registration, which then the user must provide with every request as proof of his identity. The JWT encodes within it the user’s information as well as an expiry date. Subsequently, the SpeckleServer thereafter uses this information to assign ownership information to any resource (SpeckleStream, SpeckleObject, etc.) that is created, as well as to authenticate any other incoming requests. As follows, this

---

37 To note, the authentication layer has, at the time of writing, evolved to incorporate other methods, such as Active Directory, or other, OAuth-based flows, which delegate the proof of identity to other services (e.g. internal corporate identity providers, Twitter, Github, etc.).
allows us to keep track of who is creating design information, as well as who is accessing it at the API level.

Moreover, this identification mechanism allows us to implement a permission system at the resource level. In short, this enables users to control the accessibility and visibility of the design data they are producing. For example, when streams are created, they currently default to a “public” status, which enables anyone to read them, but only their owner to write or edit them. This is similar to the “link sharing” functionality popularised by web applications such as Google Docs. Finer grained permissions can be set, enabling various levels of control, so as that some users can have read and write permissions, or only reading and commenting, or potentially fully private and accessible only to their owner. Technically, this is achieved by expanding the schema definitions of each object to include a field for storing the user’s id that created the resource (owner), an array of users that have read permissions (canRead) and write permissions (canWrite).

In short, the SpeckleServer implements a Discretionary Access Control (DAC) system. This is formally defined as a means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The discretionary aspect relates to the fact that a user, with the right set of permissions (namely write) on a given resource, can pass on that right to any other user she or he needs to. This particular pattern was chosen over the more common group- (or role-) based access control systems as it further enforces the end user to consciously share a given resource with others, and thus, give thought to the question of whom he is engaging with specifically—as opposed to simply sharing information with an abstract group of persons defined by an access level.

### 6.3.2 Clients

The authentication mechanism described above accommodates only partially the embodiment of sequentiality in a digitally enabled communicative process. This is because, in a digital design process, information is authored in more than just a verbal (or textual) way: each stakeholder, or each discipline, uses various software tools to solve parts of the assignment. For example, based on the instrumentation described so far, the research instrumentation can track that Mary, the lead architect on the project, has updated the architectural drawing plans and the massing model. Similarly, Howard, the façade engineer, has previously retrieved the information pertaining to the massing model. Nevertheless, in order to be able to trace the casual relationships between these communicative acts, and thus communicate to the actors participating in this dialogue the sequence of their informational exchanges, this information needs to be coupled with its origin as well as with where it is consumed.
As such, a new concept is introduced, that of *Clients* (full name *UserAppClient*) that track where and what information is produced or consumed and by whom. In order to do so, an instance of a *Client* object contains a reference to a *SpeckleStream* (thus answering the question of *what* information is being communicated), a reference to the *User* that created the client (answering *who* is communicating), the *role* of the client (*sender* or *receiver*), and several fields that identify *where* the information is being produced or consumed. These are (1) *documentGuid*, (2) *documentName*, (3) *documentType* and (4) *documentLocation*. These fields are self-explanatory—they help to identify the original file or model (through its name, unique identifier if present and its location on the user’s file-system) and host application (such as Rhino, Revit, etc.).

Consequently, through the implementation of application-specific clients, one can guarantee, to a certain extent, that when information is being produced and consumed, it is intrinsically tied to the specific location of these events. Furthermore, by cross-referencing clients against their *documentType*, *documentGuid* and *documentName*, one can identify casual relationships at a global level between initially disparate information exchanges. For example, *Stream A* and *Stream B* are being consumed by two individual *Clients* that, nevertheless, are identified as being in the same *Model X* as where *Stream C* originates from. While the exact nature of the relationship cannot be programmatically deduced as it is up to the end-user, this does strongly suggest that *Stream C* is dependent on *Streams A* and *B*. Furthermore, this mechanism is well suited to identify the feedback loops within the digital design process, as the process graph that can be assembled is not necessarily directed. Expanding on the example above, let us assume another model, *Model Y*, where Stream C is consumed. Model Y is also the source of Stream D, which—for the sake of brevity—is being received in the original Model X. In this scenario, what one can infer is a recursive dependency between Model X and Model Y: information being produced in X is consumed in Y, and vice-versa. A less abstract scenario can be formulated as a design-data informed conversation between a structural engineer and an architect, who iterate on a given problem’s solution based on each other’s inputs (and, by extension, expertise, viewpoints and values).

Summing up, *Clients* help us ground the informational exchanges between the actors involved in the design process with a higher degree of fidelity, and are the principal means through which one can reconstruct the sequentiality of a digitally informed design dialogue. It is important to note that the resulting process diagrams, or the communicative network that is thus documented, is not one-directional (such as a DAG) and can contain feedback loops. Furthermore, one can envision both a passive role, by which one may document the actors’ communication network and identify
bottlenecks or potential tasks that can be concurrently executed, but as well as an active role in which the network is articulated in a top-down fashion and thereafter acted out by the respective stakeholders involved in the design process.

6.4 Information Propagation

The previous chapter described the way design information is stored and retrieved and the various interactions between the persistence layer of the system and the subsequent application-specific implementations through the REST API layer exposed by the SpeckleServer. Nevertheless, these operations are silent: if someone updates a specific object, or set of objects, this act itself is not communicated; in order to verify that one has the latest information, one has to explicitly query the server for an update. Nevertheless, this does not satisfy the requirements for a communicative contract. Consequently, Speckle will leverage the virtual process diagram that can now be assembled based on the new concepts introduced above (Users and Clients) to communicate, or propagate information pertaining to the status of the data they are transacting.

In order to do so, a secondary real-time communication layer is implemented within the SpeckleServer that allows for near-instantaneous volatile message passing between one or more clients. At its core, it is based on the WebSocket (WS) protocol, which was explicitly designed for this purpose. Consequently, the SpeckleServer now exposes two different types of servers: HTTP, that enables the REST API described in the previous chapter, and a WS server, that allows clients to communicate between themselves in real-time.

Because of the stateless nature of the SpeckleServer, there is a distinct need to coordinate these messages across potentially an infinite number of instances running in parallel. This is achieved by introducing a new in-memory fast data store, namely Redis, that acts as a message buffer between the individual server instances. For example, if an instance of the SpeckleServer receives a message, it will propagate it, via Redis, to all other SpeckleServer instances.

This supplementary communication channel is used for the transmission of meta information regarding the actions undertook by a client, e.g., “the following objects were created” or “client X received the updated information”. This is achieved through a set of predefined (at the client level) real time messaging types that deal
with data creation and updating. Moreover, Speckle allows for the expansion of this protocol on a client by client basis in order to serve different needs such as automation and analysis.

The WS communication protocol implemented by the SpeckleServer is structured around specific channels, or “rooms”, that correspond to individual streams. The WS API itself contains the following operations, which clients can use:

- **Join**: A client expresses his interest to join a specific messaging room, usually pertaining to a specific SpeckleStream.
- **Leave**: The opposite operation to the above, namely leaving a SpeckleStream’s announcement room.
- **Broadcast**: A client broadcasts a message in a specific SpeckleStream’s room to all clients that are currently present in it.
- **Message**: Sends a direct message to a specific client by its _id. This does not require the client to be actively present in any SpeckleStream’s message room.
- **Ping**: Message sent by the server to all connected clients at a specific time interval to check whether they are still connected. If no response is given, after a certain set of missed “pings”, the client is removed from the server.

The actual decoding, interpretation and subsequent acting on of the information contained in the WS messages is done at a client level and is not enforced at a central level in any way. This entails an approach based on convention, rather than standardisation. As part of the existing software integrations, SpeckleRhino, SpeckleGrasshopper, SpeckleDynamo, and SpeckleThreeViewer, all the clients implement only two message types. These are:

- **global-update**: This message is broadcast in the event of an application client updating a SpeckleStream’s object list, name and any other data.
- **metadata-update**: This message is broadcast when an application client updates a SpeckleStream’s name and/or layer list.
As follows, these two message types allow for the propagation of information across the loosely coupled network of assembled clients by enabling authoring clients to notify the clients that consume information that data has changed. Conversely, rather than having to explicitly check if design data is stale, consuming clients know when their source of information has been modified and, depending on their implementation details, act accordingly. For example, the **SpeckleGrasshopper** integration defaults to pushing out data updates continuously on any change (with a maximum of one update per second). Receivers, when notified, will also immediately “refresh” and pull in the new information. Nevertheless, the online viewer, **SpeckleThreeViewer**, upon receiving a message that data is now stale, will display a notification to the user and not update instantly, thus passing the responsibility of coordination to the end user. Figure 51 exemplifies such an exchange between three different clients, A, B, and C as mediated by a **SpeckleServer**. Client A is the sender; Clients B and C can be associated with the **SpeckleThreeViewer** and **SpeckleGrasshopper**, respectively.
6.5 Case Studies

The following case studies, collected both from during the secondments with the project’s industry partners, as well as from private initiatives that customised Speckle to meet their own needs, demonstrate an enriched digital design communication process. The first leverages the transactional potential of Speckle to integrate technical (computational) actors in a digital design process. This is followed by two other examples that focus on the human aspects of a multi-disciplinary design team and their communicative process.

6.5.1 Solution Space Precomputation and Mass Customisation

By providing a digital basis on which nextness can be reconstructed in a virtual environment, one can define communicative contracts that, through more explicit messages, encode new behavioural actions for non-human (technical) actors. Speckle has been applied in this manner for enabling solution space exploration interfaces. Specifically, it was used to parallelise the pre-calculation of multi-dimensional solution spaces.

Design space exploration has been a recurring topic in computational design, visible in the late 1970s work by Robert Aish and Tom Maver with PARTIAL, a system whereby actors could evaluate their design layout based on its performance in several metrics (Aish and Fleming, 1977). Woodbury et al. define the term design space as a “network structure of related designs” (2005) and its exploration as “a guided movement through a space of possibilities” (1999, 2000). There is a wide body of research focusing on the interaction between designers and the solution space they navigate with the help of computers (Burrow and Woodbury, 1999; Woodbury et al., 2000; Woodbury, 1991; Woodbury and Burrow, 2006a, 2006b, 2003), which partly predates the emergence of popular parametric design tools, such as Generative Components, Grasshopper and Dynamo. As a result of this new generation of software, designers stopped creating static solutions and moved on to encoding building logic into parametric models. Thus, from solution spaces that were defined by a low, human-producible number of options, the current status quo is that in which one has the potential to evaluate a continuum of data points, each representing a unique design variation. Consequently, when employing parametric modelling, designers both define and subsequently explore design spaces. Simplifying, the process largely consists of cycles of generation and evaluation: a designer sets the input parameters, generates the result, and finally evaluates it according to both objective factors as well as subjective criteria.

The process of generating discrete instances from a given design space, as defined
through a parametric model, can be automated by the pre-calculation of a discrete set of design instances. Current parametric modellers, such as Grasshopper or Dynamo, do not allow for displaying multiple states simultaneously, which makes a comparative analysis difficult. By leveraging the technical infrastructure defined previously, one can treat a parametric model as a simple technical actor and, through a custom protocol composed of several new client-interpretable messages, create a workflow that allows for a concurrent calculation of a solution space. The methods presented here revolve around “brute-force” sampling, and serve only to demonstrate a technical contribution.

In more words, one initially defines the degrees of freedom of the parametric model that one wants to pre-calculate, as well as its evaluation criteria, if present. The geometry is curated from throughout the model by being transmitted into one original SpeckleStream A; the input variables (sliders) are flagged by prefixing their name with the SPK_IN string, and the output variables (besides geometry, and embodied in Panels) with SPK_OUT. The accuracy and type of the sliders will also determine at what interval they will be sampled: for example, an integer slider with a minimum value of 0 and a maximum value of 10 will contribute 11 data points. To enable pre-calculation, the SpeckleStream sender component must signalise this by setting the option of “Enable Compute” to true. If the exact same definition is opened multiple times on multiple computers, all of them will participate in the computation process. Subsequently, with the aid of an online interface, a user can start the precomputation process, which will distribute equal amounts of data points to each compute-enabled client. The process is as follows:

- The online client W accessing Stream A broadcasts a custom “compute-enabled” message in Stream A’s WS message room.
- Clients X, Y, and Z receive that message and send a direct “compute-enabled-response” message to W, which also contains each compute-ready client’s parametric model hash.
- Online client W waits for a predefined interval of time (2 seconds) for the above responses to arrive. Afterwards it compares the hashes of each response to check that they do not differ (as this would imply a different parametric model).
- Subsequently, online client W will create the permutation set of the solution

---

38 This matter has been researched extensively in the field of design space exploration. Brander et al. have studied how design professionals generate and evaluate designs (2014). Other literature looks at various sampling methods and their implications on the act of exploration (Burrow and Woodbury, 1999; Woodbury et al., 2000; Woodbury and Burrow, 2006).
space based on the input provided by X, Y and Z.

- Afterwards, it will then start sending “compute-request” messages containing one different set item to each of the three clients.
- A client, upon receiving a “compute-request” message will calculate that specific instance of the solution space and save it in the database as a new *SpeckleStream*. Once this is done, it sends a “compute-result” message.

Parametric modelling software, such as Grasshopper, is not designed to take advantage of modern multi-core processor architectures, as its main concern is to calculate fast only *one* instance of the model at a time. Consequently, parallelising this process reveals performance improvements even on a single modern laptop, with each extra instance of Grasshopper being open for the generation of a solution space contributing an almost two-fold increase in speed. This opened up new avenues in digitally enabling participatory design (Figure 52). Parametric models, through their inherent flexibility, can be used as a negotiation, or communication tool, by designers. Previously, this required specialised equipment and software that was thus limiting the outreach and engagement of a parametric model: one needed to have access to CAD software in order to use it (Knecht et al., 2019).

![Figure 52: Screenshot of the online interface developed for citizen engagement, based on a pre-computed parametric model. From Knecht et al., 2019.](image)

Another example of the potential to integrate technical actors arose from the field of bespoke manufacturing. Two separate companies, NStance and Simple Wood Goods, used Speckle as the data interface between their customers, their proprietary design
logic and their manufacturing process: the former towards 3d printing, laser cutting or metal casting of bespoke jewellery (Figure 54), whereas the latter on CNC milling and assembling of a parametrically defined table (Figure 53).

Figure 53: Simple Wood Goods example application: web interface (left), customised real product (right). Image rights: Matthew Swaidan, 2018.

Figure 54: NStance web application (left), and corresponding ordered product (right). Image rights: nstance.ro, 2018.
The resulting process is equivalent in both cases. As the diagram above shows, it starts when a person accesses the online customisation interface developed (Figure 55). Initially, there are a series of exchanges, or customisation cycles, during which the user configures the end product to suit his needs. Both NStance and Simple Wood Goods used a separate parametric model specifically geared towards visualisation. Subsequently, when an order is placed, it is thereafter picked up and processed through a much more detailed parametric model that is, ultimately, producing the information needed for manufacturing. This amounted to 2D dimensional laser-cutting files in the case of NStance, grouped together with a labelling and colouring system; in the case of Simple Wood Goods, the Grasshopper definition would ultimately produce machine code for a CNC router. Followingly, the product would be manufactured and shipped to the end-user.

In the case studies presented above there was always a human involved in the process (in the manufacturing step). Nevertheless, using the same programmatic base used
to enforce a social communication contract, one can include technical actors in the “conversation” that execute codified tasks, automating parts of the design and manufacturing process, through the elaboration of custom, machine-to-machine, protocols.

6.5.2 Design and Visualisation Workflow

Propagating information across a loosely coupled design process is crucial for the correct functioning a multi-role team. During the industrial secondment within HENN Berlin, a workflow (and data sharing patterns) revolving around the design and delivery of an early stage architectural competition has been observed. The actor network was composed of: (1) several architects which had the conceptual and design responsibility of the overall project (the design team, with its own internal leader—the project manager); (2) the visualisation team, which was responsible for the delivery of the project’s graphical images and diagrams (which was simultaneously helping several other projects); and (3) the firm’s senior partners who would oversee the progress of the project on a weekly or bi-weekly basis.

The habitual interactions revolved around weekly cycles determined by the senior partners’ reviews. For these half-day long meetings, the design team would summarise its progress and, together with the visualisation team, produce graphical output that would then later be used as a basis for the presentation and discussion with the senior partners. Most of the design work happened in Rhinoceros (sometimes aided by parametric modelling or environmental analysis in Grasshopper). Based on these exact same 3D models, the visualisation team would then produce renderings and diagrams for presentation purposes.

This approach, while extremely efficient, was nevertheless susceptible to a breakdown in communication. For example, there were several cases in which the renderings were produced based on stale model data, fact which then led to an unproductive review meeting in which the project lead responsible and his team were “talking” something different than they were “showing”, much to the distress of all the parties involved. The reasons behind this failure can be attributed to many factors, some of which have to do with the digital infrastructure used for communication purposes. The file-based approach used (shared network drive) does not propagate information to the responsible parties; there is no mechanism by which the interest and relevancy of specific design data can be tracked and acted upon.

Within this context, the SpeckleRhino integration was used to trial an alternative approach. The design team split up its working models into several SpeckleStreams
which were then read by the visualisation team. Data was distributed according to common-sense patterns that were specifically helpful for this particular workflow, namely by individual building and their respective façades, as well as several key elements that represented central project cornerstones, such as the lobby of the central building and several key site-based landscape entrances. Accordingly, the main difference between approaches that the teams reported was a certain “peace of mind” that came because of the assurance that whenever they would need to operate on specific design data, they could rest assure that it is “up-to-date” and not stale. For example, a member of the rendering team gave an account of how, whilst preparing his visualisation model for a render, he could instantly see that the façade of the building changed as the design team were doing last minute changes to it. Consequently, he immediately noticed that certain assets (people, cars) were now starting to obstruct a key element of the entrance to the building, and could re-adjust his model so as to prevent this, thus “not wasting two hours” setting up a bad rendering scene.

Similar workflows spread internally, within the project’s core team for both coordination roles amongst the various loosely coupled parts of the overall design (massing, landscaping, parking, structure, facades, etc.). The main advantages cited by the “early adopters” was the absence of costly and “clunky” imports and the friction associated with sharing whole files over a network drive, no more “copy-paste” operations from one Rhino model to another and the fact that the process was “faster”. A widely used workflow was the possibility to do on-the-fly reviews, with senior partners being able to access the in-progress designs via the online 3D viewer (SpeckleThreeViewer) and provide commentary and guidance even while travelling from their mobile devices. This revealed different interactional requirements of the digital design process (i.e., comments, graphical notes, sketching on a 3D model, etc.), which shall be discussed towards the end of this chapter. To conclude, this simple case study validated the relevancy of a digital communication process that not only updates design data, but goes one step further and notifies and propagates the relevant information to the relevant parties in an accessible manner based on the situation at hand.

### 6.5.3 Design and Façade Engineering

Another example of multi-disciplinary collaboration where Speckle was used as the vehicle for propagating information across a diverse range of actors was in the design of a central, “iconic” element of an airport expansion. This element consisted of a 90 meter wide circular mixed cable-net and circular beam structure that resembled a
“vortex”, and it supported upwards of 1000 glass panel elements, each unique in its size and material type. Unfortunately, at the time of writing, the project is restricted by confidentiality requirements, which prevent us from providing further, more specific, details.

The team responsible for its design was spread out across three companies, namely “The Architect”, “The Structural Engineers”, and the “Façade Engineers”. Specifically, while the general outlines of the design were fixed—specifically the “vortex” metaphor, and the boundaries of the element—the actual rationalisation of the concept into a buildable object left multiple degrees of freedom that needed to be explored in order to reach the best compromise in terms of budget, aesthetics, performance and ease of construction.

At the initiative of the façade engineers, the project team devised a workflow that would allow them to rapidly iterate through various options while simultaneously informing all of the participants of the outcome. This workflow was thereafter implemented with Speckle. Specifically, the team separated the informational concerns into three main speckle streams based on the disciplines involved, namely (1) design, (2) façade engineering and (3) structural engineering. The design stream informed both the facades as well as the structures stream. The structures and facades streams were tightly intertwined in a feedback loop, as they exchanged much technical information throughout the course of the project. While at the beginning of the design process, both the structure stream as well as the façade stream were fed back to the design team, eventually, the façade stream evolved into a centralised data source for curated information that the architectural design team could thereafter act upon.

As the project progressed, each discipline data stream started being informed by and informing several other models through secondary speckle streams. These pertained to various sub-tasks that, as the project progressed, could no longer be held together into one model. For example, (3) structural engineering was having three separate child models, namely a (3.1) load analysis stream, (3.2) movement accommodation and (3.3) secondary structure design stream. (2) Façade engineering, besides curating the input from (3) structural engineering, was also being informed by streams referring to (2.1) cable net movement checks and (2.2) drainage analysis (Figure 56).
The feedback received was mostly technical in nature—specific issues with dealing with complex NURBS-based geometry elements—but contained interesting reactions to communicative aspects of the process. Specifically, the engineers and designers were happy with the “responsiveness” of the process, nevertheless they requested more refined means of controlling the data updates so as to not “flood” the system with arbitrary data (this was implemented). Several positive remarks were addressing the fact that the author of a specific SpeckleStream could see in the online interface who was consuming it, or, in other words, where that information is received. Thus, the façade engineer working on the drainage analysis noticed that his analysis results were being consumed by a colleague in the structural engineering team, who was using that information to inform the size the secondary beam structure. Nevertheless, this view did not satisfy the needs of the two project managers from the design and façades teams: they expressed the need to be able to have an overall view of all communication network, similar to the diagram above, in order to coordinate the efforts of their team members.

Similarly to HENN, the project managers’ feedback also described another set of interactional requirements coming from the coordination meetings. Specifically, they expressed the need to be able to textually define issues on specific elements, or have various analysis views in the online interface that would allow them to point out specific evidence to support their arguments while discussing the compromises in the design of the “vortex”.

Lastly, by being able to exchange design and analysis information in a fast and flexible manner, the actors involved in the process were able to iterate and evolve the topological structure of their communicative network towards an “optimal” state.
that matched, as the design progressed, new instances of problem definition and subsequent resolution. In other words, they were able to support the dynamic management process of the wicked problem they were tasked with (the design of a complex structure).

6.5.4 Summary

The three case studies presented in this section are summarised in the table below (Figure 57). The key learnings are expressed in terms of positive observations (+), and ones found lacking (-).

<table>
<thead>
<tr>
<th>Case study</th>
<th>Actors</th>
<th>Summary</th>
</tr>
</thead>
</table>
| 6.5.1      | (a) End-users  
(b) Product Designers  
(c) Technical Actors | (+) Fast transactions between (a) end-users and (c) technical actors enabled online customisation cycles. Existing file-based approaches would not be suitable due to their high overhead.  
(+ ) Custom protocol definitions enable more than simple “update” notifications between participants, and open up the design and manufacturing process to automation.  
(-) Product designers were partially dissatisfied with the limitations that the online medium offered in terms of presentation. |
| 6.5.2      | (a) Architects  
(b) Visualisation Artists  
(c) Senior decision makers | (+) While file-based approaches were successfully used in the past, Speckle allowed (a) architects and (b) visualisation artists to coordinate better.  
(+ ) Reliably being able to observe changes in real-time allowed (b) to feel in control of the process, and pace around (a)'s creative output and instructions. |
| 6.5.3      | (a) Architects  
(b) Structural Engineers  
(c) Façade Engineers | (+) All stakeholders (a, b, and c) were pleased with the responsiveness of the system and the ability to organically structure their communication process.  
(-) The product managers from (c) highlighted the need for a more comprehensive management interface to track the communicative exchanges between the team’s members. |

Figure 57: Data transaction case studies summary table.

The case studies presented above involved a diverse number of stakeholders from the design process, such as engineers, architects and product designers which have used Speckle in order to improve their workflows. The existing implementation, when applied on a larger scale, was found lacking in this regard as project managers could not have an overview of the dialogue being undertaken: while being able to trace source and dependents on a single stream basis, on a project basis, involving multiple streams, this was not possible.

The need for an “global” overview arises from the fact that communication in the
design process is a dynamic process, as also evidenced by the last two case studies. These highlighted the organically evolving network of data sharing, tending to its maximum relevant state, as Sperber and Wilson would put it (2008). This was seen by the participants as a welcome development. Notwithstanding the limitation mentioned above, the fact that information is not simply mechanically transmitted, but that transactions between the stakeholders tie them into tangible communicative contracts, was beneficial in comparison with the existing file-based approaches that do not establish these relationships.

6.6 Conclusion

This chapter looked at enabling efficient and meaningful design data transactions from a technical and social point of view. All communication processes rely on a social contract among participants, two key properties of which are nextness and sequentiality. Nextness is the property of utterances to be adjacent to each other so as to maintain relevance—essentially, their speed and frequency. Sequentiality refers to the logical order in which utterances are exchanged, and ensures the overall coherence of a conversation. In a digital environment, as opposed to a human, face-to-face one, these two key perquisites of communication are not directly enforced. Subsequently, the issue behind the lack of nextness in digital design communication was analysed. Second, the investigation focused on how the sequentiality of digital information exchanges between actors can be expressed and revealed to the participants in order to reduce the intensive coordination requirements resulting from detaching the information being transmitted from its source.

In a digital environment, nextness, or adjacency in between informational exchanges, is directly dependent on the size of the data payloads that are transacted. A file or model-based approach imposes a high cost, thus reducing the frequency with which stakeholders “communicate”. In order to increase the speed with which these transactions happen, Section 6.2, Optimising Transactions: Differential Updates puts forward a client-side differential compression algorithm operating at the level of design data object sets that leverages the approach to object identity and immutability adopted in the previous two experiments to incrementally create a shared informational context.

This is thereafter used to reduce information redundancy and consequently the amount of information being transferred. Similarly to natural dialogue, where participants rely on their shared context in order to increase the relevancy of their exchanges, so does Speckle leverage a technically created context for speeding up design data transactions. In regards with sequentiality, the mechanism for tracking
the interdependency of the communicative exchanges in a digital workflow described in \textit{Section 6.3, Assembling the Network} was implemented. This was achieved by separating the storage from the actual message transaction system, thus allowing for fast, spontaneous messaging between end clients as well as reliable transmission of very large data sets. The actual tying in of data, author and source is achieved through a new data structure (clients) that keeps track of these aspects, complemented by a discretionary access control policy, which authenticates every action against a specific end user’s credentials (\textit{Section 6.3.1, Users} and \textit{Section 6.3.2, Clients}).

These technical solutions were validated in a technical environment, and empirically assessed in terms of performance. Furthermore, several applied case-studies, emerged from the usage of Speckle within the living laboratory context of this project, have served as the basis for a qualitative analysis. Specifically, the findings of this chapter can be summarised as follows:

- The transaction size can be decoupled from the size of the model, thus allowing for much smaller, and faster, change-dependent digital design data exchanges (\textit{Section 6.2}). In more detail, the empirical observations show that:
  - Differential updates, on average, are half the size of an equivalent file-based transaction. Furthermore, based on empirical observations, by employing this method, transactions can be as much as five times faster (by virtue of being five times leaner in size) (\textit{Section 6.2.2}).
  - Nevertheless, differential updates have no theoretical upper limit when it comes to potential efficiency gains, and in their worst-case scenario they are equivalent to a file-based methodology (\textit{Section 6.2.3}).
- Informing users about who is dependent on their data, as well as when the data they are depending on, coming from a different person, has become stale, establishes a measure of productive sequentiality which is sufficient for individual tasks (\textit{Section 6.5.2}).
- Because communicative networks can be assembled organically (as previously discussed in \textit{Section 5.6} of the previous chapter), this potentially leads to a disorderly process. The case study presented in \textit{Section 6.5.3} has consequently revealed this shortcoming and identified the need of stakeholders in coordination roles (project managers) for a project-wide overview of the activity/process network.
7. Discussion
The main investigation areas presented in the three previous chapters, seen together, sketch and evaluate an alternative to the way digital design information, or data, is being represented, classified and transacted within the AEC industry. Being an applied research project, with heavy input and demands from the industry partners, the software instrumentation developed (Speckle), originally intended purely as support for this research, evolved beyond its intended scope. This is in part due to the technical affordances of the platform, and the communicative processes it enables, but as well due to political factors that normally lie outside a scientific analysis. While the former have been discussed at length in the previous chapters, this section shall now endeavour to tie in both political and technical aspects of the quintessentially wicked problem of digital design communication. As Bruno Latour and other thinkers have shown, scientific pursuit cannot be fully untangled from its social and political context (Harman, 2009; Latour, 2007).

7.1 Schemas and Standards

Chapter 4 questioned what the best way to represent digital design data is in order to support the needed processes of ontological revision that underpin social communication processes. Specifically, the investigation looked at whether an approach based on a composable and incomplete object model can better serve these needs, as opposed to an ontologically complete standard. The results of this investigation (Section 4.3) showed that, if given the possibility and backed by adequate tooling, users will develop their own higher-level digital design ontological definitions in a way that allows them to structure and evolve their communication in a more lean and efficient way—echoing Grice’s maxim of quantity (Grice, 1991), as well as Sperber and Wilson’s relevancy principle by which information tends to follow a path of least “effort” (Wilson and Sperber, 2008). Going even further, Section 4.2 has shown the potential for “backwards compatibility” with existing standards, by programmatically matching the flexibility of a composable lower-level schema with existing object models (BHoM, and, partially, IFC).

The main implication of this result is that, with regards to digital data representation in AEC, the industry has, so far, operated at an inappropriate level of abstraction. If

39 At the time of writing, Speckle continues to evolve and grow its community of contributors and adopters.
a composable object model approach can offer space for both flexibility as well as consistency, it essentially has all of the advantages and none of the drawbacks when compared with a rigid standard that has the ambition to be ontologically complete.

To unpack the previous statement, one needs to take a wider view of the question of standards. Any communication system will, without a doubt, impose a certain formalised approach. For example, in the case of human communication, this takes the shape of a given alphabet and language, which, in turn are further enriched by a shared social context. In the case of digital communication, due to the unforgiving technical nature of the transmission medium, there are a number of standards that operate at various levels of abstraction, from the low level error correction mechanisms of binary information packets to increasingly more human-understandable layers of information structure and semantics. For example, the internet, as it is known today, is built on top of many layered standards: at the transport level, providing host-to-host communication services, one finds TCP, UDP, QUICK; at the application layer, standardising communication on top of the transport standards above, is where the SSH, FTP, HTTP, SSL/TLS protocols operate. On top of these layers, one finds the technologies that operate at the content level, which describe the human- and machine-readable representational aspects of information (HTML, CSS). Without these standards and protocols—essentially mutually agreed upon conventions—whenever one would want to communicate something digitally, one would need to fall back on 1’s and 0’s and have to re-implement error correction mechanisms from scratch.

The AEC industries have only one such standard, namely the Industry Foundation Classes (IFC). The design ontology described by the IFC standard is one that does not lend itself to modularisation and, subsequently, exchange and revision by actors from the outside. A naive comparison that reveals the scale of the problem is that between IFC and HTML, the markup language that structures all web-based documents. The number of specified HTML standard elements is around 100; the number of IFC elements, at a cursory glance, is higher than 1000. HTML is a low-level standard whose primary expressivity results from its composability and its ability to interact with other systems, purpose-built for different needs: they can be combined in various ways so as to create novel higher-level objects that serve an enormous amount of functions and shape information into meaning both for non-human (machine-readable) and human actors (through rendering). The original set of

---

40 Visual appearance (styling) is controlled primarily via CSS; interactivity and real time logic is provided through JavaScript.
SpeckleObjects (presented in Section 4.1) was developed with this in mind and aimed at offering a flexible base of CAD “atoms” that could be combined by end users into higher-level logical objects that encapsulate richer meaning.

The findings from Section 4.2.2 have shown that IFC’s complexity translates as well in a large object size, approximately 12 times larger than other leaner pre-defined schemas. Following this, this builds a convincing argument towards the fact that IFC, given its sheer size, scope and implementation directives, does not allow or facilitate a communicant to respect Grice’s maxim of quantity. In other words, it operates at a much too specific level of abstraction without allowing for improvisation and customisation. Its rigidity leads also to a perceived lack of expressivity: end users and developers alike are not tempted to experiment with it.

On the other hand, a small and flexible standard lends itself easily to “hacking”. In Section 4.2.3, one of the most striking findings was that one in five base Speckle objects, out of the 16 million analysed, were enriched by end-users, strongly suggesting that this approach was, at least in terms of popularity, well received. Compared to the tree depth of an object derived from a pre-defined schema, end-user generated objects’ hierarchy was mostly flat (an average of 1.97 levels vs. an average of 3.43), thus further supporting an argument against complexity.

Moreover, end-user defined standards can leverage the process of self-contained “emergence” at a higher level of abstraction, as well as become a vehicle for positive network effects outside its original remit by cross-coupling with other technological stacks for different purposes. To illustrate the process of self-contained emergence, several metaphors come to mind. The most naïve, human language, illustrates this: given a finite amount of building blocks (words), a wealth of knowledge and expressivity can be articulated beyond the simple sum of meaning that the words themselves carry. Conway’s famous Game of Life is, perhaps, a better example: given a finite number of states and a finite number of simple rules, the state machine exhibits emergent behaviour (rich interactions that could have not been predicted, or instrumented for).

The latter implication, namely the ability to easily cross-couple with other technological stacks, is extremely relevant in today’s environment. By being understandable and having a manageable cognitive entry barrier, a small composable object model lends itself easily to the digital ambitions of AEC companies. To expand on this, the AEC sector is currently undergoing a “digital transformation”; this has materialised as programmes of “digital transformation” which are specific to each company undergoing one. The needs that arise from these changes put an onerous pressure on how the huge amount of digital design data that these stakeholders
produce is used to drive insights from existing projects (that feed back into client and
design-facing functions, such as bids) as well as various automation tasks. The source
is still design data itself, nevertheless it needs to be ontologically redefined and
grouped under various other definitions to serve different purposes. For example,
Speckle has been used to drive MEP report generation, embedded carbon
calculations, and other automation tasks, all linked with (or based on) original design
data. The ontological transformations between the various representations of data
(for example, from an architectural definition of a room to a PDF report containing
air flow requirements and mechanical equipment provisioning) are not trivial, and
required the use of various other technological stacks to process and render. Further
research is needed in this area to better identify the various layers of abstraction, their
interaction with other technologies, and how the mixture can serve the needs of
industry stakeholders in their pursuit for a more integrated digital built environment.

7.2 Classification and Curation

One of the limitations of this “fluid” approach to standards is the potential for data
loss. Digital design information, when contextualised in different authoring software,
mutes to better fit the available native definitions at hand. For example, a beam can
be (1) originally defined as a line with custom properties in Rhino, (2) then become a
three-dimensional extrusion with associated parameters in Revit, (3) thereafter
morph into an analytical one-dimensional object in a structural analysis package, and
finally, (4) a row in an spreadsheet or construction management software.
Throughout all these stages, there is no singular object. While at certain stages of this
ontological transition process, for example (1) and (3), there are shared geometrical
definitions, consistency cannot be guaranteed throughout its whole entirety. This can
be traced to the way knowledge is developed and encapsulated in professions: the
definition of beam will be different for an architect, a structural engineer or an asset
manager. As Abbott would put it, each, when presented with information about a
beam, will reshape it to fit the internal standards of their discipline and its body of

Consequently, Chapter 5 looked at what is the best way of persisting, or storing, and
classifying digital design information for communicative purposes. This was done by
mitigating, or managing, the limitation above, as well as by comparing the existing
file-based paradigm with an object-centred storage approach. From a technical point
of view, Speckle demonstrates that by introducing object fingerprinting and
immutability at the storage layer, data loss is avoided. Furthermore, the analysis of
the instrumentation and its application in practice within the wider industry network
show that a curated, object-centric approach to data exchange is better suited for
digital design communication rather than a file-centric approach in which data is shared “in bulk”. These two approaches, rather than excluding each other, can be seen as complementary: the former file-based approach is well suited for design modelling purposes and is deeply rooted in the way AEC authoring software works, while the latter better facilitates data informed dialogue, and allows for the natural evolution of optimal communication channels.

A major implication stems from the fact that, from a technical point of view, both object- and file-based approaches are similar, yet they diverge in the way they can be applied and the nature of the communicative exchanges they foster. At a diagrammatic level, in both a file-centric exchange methodology as well as an object-centric implementation, information storage is centralised. In the former, this amounts to the “one model” strategy, by which all subdisciplines contribute to, and coordinate with, a unique federated model which acts as a single source of truth. The counter approach is based on a minimal user defined pre-classification, in which objects exist independently of the groups they form; nevertheless, information is still stored centrally within a database. This allows for the sub-classification of a given design into an infinite number of “models” through the mechanism of curating objects into as many groups as needed, based on the criteria that emerge as useful.

The empirical analysis from Section 5.7.3 shows that, in the context of two models, one staged as an example, and one from a real-life project, this leads to important storage savings, which in turn translate to a better ease of use. Subsequently, this leads to an evolving emergence of lean communication channels between the actors involved in the design process, based on based on exchange requirements defined on the fly. These technical nuances, when applied in practice, lead to a divergence in the resulting communicative process: one results in a rigid process that draws clear lines between formal and informal communication links, and discourages the latter; and the other allows for natural communication patterns to emerge and evolve throughout time.

To exemplify, an object-centred approach enables one given design model to be separated into multiple independent classifications for different purposes. One such criteria of classification can be based on functional differentiation of building elements: structure, facade, furnishings, etc.; a different one can be based on the building’s levels, namely ground floor, first floor, second floor, etc. It is not possible to pre-determine what the productive categorisation of design data will be for a given project or building; even more, there probably isn’t one singular way of slicing the design information produced during a building project. In this respect, because an object-centred approach provides granular access to individual objects, data can be
classified around ad-hoc, arbitrarily defined queries or criteria. These can be defined on the basis of specific stakeholder interests, as they evolve throughout the project. Furthermore, due to design data no longer residing exclusively in traditional files (that needs a specific software in order to be decoded) its outreach, accessibility and transparency is greatly increased. As an illustration, it can be both visualised in a standard browser on mobile devices, tablets and personal computers, as well as imported into widely used tabular software, such as Excel. Moreover, because design data is presented in a self-describing format (JSON) and made accessible through a clear and easy to reason about API, creating new augmented workflows and applications by expert users (or developers) is possible without prior knowledge of the whole system, thus allowing for tailored computational tools catering to tailored needs.

Informational waste is a crucial problem with regards to communication: it ends up being shared without a specific intent, or reason. Thus, in the words of Garfinkel, “noise is introduced in the system” (Garfinkel and Rawls, 2006), and subsequently leads to higher communicative costs and reduced informational relevance. Countering this trend, the curatorial data sharing methodology introduced Chapter 5 encourages a certain relevancy check on behalf of the data producers: one is required to have a reason to communicate. Both the recipient and the sender need to collaborate towards and negotiate what data is required of them and for what purposes through dialogue as well as assumption testing of what each other needs are in order to achieve a specific task—therefore the resulting information being exchanged has a specific intent behind it. By ensuring that every “utterance”, or every piece of information that is shared has, to a certain extent, a specific reason for being shared, the risk of creating “disembodied” data (Tsoukas, 1997; Giddens, 1991) is minimised. In other words, data has a given positive effect for the receiver.

The findings in Section 5.7.1 and Section 5.7.2 support the curatorial approach to data exchange described above. Usage data gathered through Speckle revealed that, on average, a single file gives birth to two or more sources of information; similarly, on average, a file can re-combine two different sub-classifications created elsewhere. Overall, they show that data is productively being consumed by end-users—with a source to receiver ratio of 2.26; in other words, for a single emitter, there are, on average, two or more downstream dependencies. One major implication is that, so far, current information exchange practices, based on the federated model as a single source of truth, needlessly impose a technical model of communication (that underpins their software implementations) on what is, in essence, a psycho-social process; the equivalence between how software communication works (Figure 58) and how social communication should be articulated is false.
Figure 58: Above: Similar technical code models of communication enabling two different processes in the context of digital design communication. Below: relationship between the communication channels and their supportive (centralised) technical embodiment: multiple users engaged in a digital communication process enabled by one central server.
The picture that the usage data from Speckle painted is one in which end-users assemble their own federated models on a need-by basis, by curating the necessary information for a given task. Subsequently, the clear dividing lines between “formal” and “informal” exchanges, with the latter discouraged from happening, are misguided: formal communication exchanges need to organically evolve from informal ones. Reciprocally, “informal” exchanges should be able to be bootstrapped from existing “formal” ones.

File-based collaboration practices, coupled with current methodologies, enforce formal data exchange patterns, and do not satisfy the need for design data informed dialogue and cooperation; moreover, they act against it by creating disembodied information with decreased relevancy due to lower generation of positive cognitive effects and increased processing effort on the side of the receiver. Nevertheless, the technical, centralised, model of digital communication can be made to support a naturally evolving process of informational exchange which, in turn, allows for the creation of shared understanding and meaning amongst the various actors involved in the design process.

7.3 Digital Transactions & Communication

One of the main limitations with digital dialogue comes from the fact that the medium does not convey the sequentiality of information exchanges: unlike face-to-face conversations, digital data transactions are asynchronous and, often, may happen in parallel between multiple actors. A second limitation pertains to the fact that the actual transaction sizes, especially in a file-based process, can be extremely large and thus compromise another key ingredient of communication, namely nextness, or immediacy.

To elaborate, communicative acts in various professional domains entail different social contracts. In the case of online textual exchanges between people, this difference manifests itself as a demarcation between informal (such as Messenger, WhatsApp, etc.) and workplace (or business) chat applications, such as Slack. The former allows for quick “banter”-like dialogue to happen, and attempts to enrich the digital conversation with various ways of expressing human emotion; the latter, while still allowing for a certain conversational playfulness, is geared towards organising discussions around assignments and topics, creating a conversation record that can be audited, and allowing for various digital documents to be exchanged between users—essentially, attempting to enhance productivity. In the case of collaborative programming, the digital platforms that emerged have different mechanisms supporting a productive workflow. For example, git (and Github, an online collaboration platform based on git), positions itself as version control systems
that allow multiple persons to edit files (usually code) in distributed, non-linear workflows. Other platforms, geared towards project management and document-based collaboration, emphasise scheduling features, client interaction management, etc. Nevertheless, in the context of digital design, current file-centric collaboration methods introduce two problems. First, because of their technical foundation relying on sharing models in their entirety, data exchanges are slow and, consequently, far spread apart, thus compromising their nextness. Secondly, because the information being shared is detached from its source, and author, the interdependency between the exchanges is lost. This compromises another key aspect of (social) communicative contracts, namely that of sequentiality.

Chapter 6 shows how these limitations can be partially mitigated. With regards to nextness, by introducing a differential compression algorithm, the size of transactions for both receiver and sender is minimised by transmitting only the changes between updates. Compared to a file-based collaboration process, the analysis in Section 6.2.2 shows that the size of transactions is greatly reduced, with observed performance being twice as high (from the point of view of amount of transacted data, and associated time). As discussed in Section 6.2.3, where the theoretical limit of this approach was examined, a file-based approach performs, regardless of the nature of the change, in a worst-case scenario, whereas differential updates can be situated along a gradient spanning from a worst-case (equivalent to existing methods) to a best-case scenario in which there is no upper bound (if the change is small enough in comparison to the whole). Keeping transactions “as close” to each other as possible allows for a much higher frequency of digital exchanges between communicants.

Section 6.3 describes how sequentiality is (re-)established by tracking the interdependency of the communicative exchanges between the participating stakeholders. This is achieved by grouping together the data, its source software application and originating file, and its owner in what de Vries calls an activity network diagram (de Vries, 1995; de Vries and Somers, 1995).

The case studies presented have shown that exposing this network for each individual data source was successful in giving design stakeholders a more grounded view of the communicative process they are playing a part in, especially for persons involved on specific sub-tasks of a project. Nevertheless, the case studies also showed the need for a holistic overview that would allow, for example, a project manager to carry out his coordination duties that a larger assignment require. In this regard, tracking individual sources of data was not enough. However, the data produced by the proposed implementation is sufficient to recreate an activity network diagram. As part of an InnovateUK grant, research is currently being undertaken on the best
way to expose the communicative network in a meaningful manner to end-users. This includes investigations on how to best group, categorise and explore it: around end-users and their roles, around files and applications (Figure 59), around a historical timeline (Figure 60), etc.

Furthermore, while the descriptive nature of visualising sequentiality in a digital communication process is important, its normative potential is also evident: industry feedback, as well as previous scholarly work from the field of product & process management, suggest that an activity network diagram could be created in advance, and then enacted during the design process—a reversed approach to its original goal, nevertheless useful for bootstrapping a known process.

Figure 59: Activity network graph generated for a set of data streams from a live project. Left: full graph; middle: grouped by users; right: grouped by documents. Image credits: Paul Poinet, research done at UCL BCPM under the AEC Deltas InnovateUK grant.

Figure 60: Different activity network, with icons symbolising different software. Image credits: Paul Poinet, research done at UCL BCPM under the AEC Deltas InnovateUK grant.
Furthermore, it is important to note that, while tracking information flow and communicative digital exchanges captures a big part of the design processes, the AEC industry deals with, equally important, processes of material flows. In the context of construction, site planning, assembly, casting, etc. represent equally important dynamic aspects that need to be taken into account.

Summing up, the analysis in Chapter 6 on transactional requirements of digital communication in the AEC industries was restricted to just two fundamental aspects, nextness and sequentiality. Nevertheless, these are not the only aspects shaping it. Further research is needed to understand users’ interactions with such a system, as well as the communicative topologies that emerge, and finally, identify all the relevant informational and material flows.

In summary, both nextness and sequentiality can be enforced (or enabled) in a digital design environment. These transactional requirements of communicative acts have been so far largely ignored in current tooling. Their inadequate implementation has shaped a slow-paced context of data exchange that is not conducive for the way actual design tasks are being resolved: the frequency of data transactions is directly linked to efficiency of communication, especially in cross-disciplinary environments. Nevertheless, the limitations coming from digital design tasks—large datasets, diverse and spread out teams, etc.—can be mitigated by reducing data transaction costs and keeping information attached to its author and source, thus allowing for the natural emergence of nextness and sequentiality among the data exchanges between the stakeholders involved in the design process.

### 7.4 Data Ownership

Going beyond the theoretical and technical analysis presented in this thesis, the instrumentation developed to support this research has, through its exposure and adoption in the industry, revealed a strong political dimension of the wicked problem of digital design communication. Specifically, this problem revolves around data ownership and, to a lesser extent, accountability and transparency of the software used. Speckle has been developed and released from the start as an open source project licensed under the liberal MIT license, and all source code is freely available online together with its historical evolution (see Appendix A, Tooling). Unlike other commercially available solutions, users are always in control of where their data resides, and they retain full rights to it.
Data ownership has always been a contentious issue in the digital age. Nevertheless, in recent times, it has come to the forefront of wider public attention due to several events, such as the scandals revolving around mis-use of user information by Facebook, the new EU regulations around data privacy (GDPR), etc. Within AEC, projects usually come with complex requirements around data ownership, safety and various other confidentiality agreements. Existing software vendors that service the AEC industry are thus finding themselves in a conflicting position. Their commercial interests dictate that they do not relinquish control of the wealth of data being produced during the design process; interoperability is a direct threat to the ecosystem of tools that they have built and that drive their revenue. Moreover, new commercial ventures that have attempted (and are attempting) to cater to the interoperability needs of the industry must, by virtue of business common-sense, stay in control of their users’ data in order to charge for it.

Furthermore, the problem is compounded by the fact that current “formal” methodologies—sometimes enforced via governmental mandates, as in the case of the United Kingdom—are adding an extra layer of complexity. The result is an industry where data flows are stifled, and thus the benefits of digital transformation are reduced. Nevertheless, Speckle was written from the start with accessibility in mind, both at an end-user level, but as well at a technical, deployment level. At the deployment level, this translates into the fact that users are encouraged to deploy their own instances of the Speckle server based on their particular needs: at the time of writing, there are more than 40 Speckle servers in operation world-wide.

Industry feedback, informally collated from the living laboratory, reveals several deployment scenarios: (1) internal, local deployments, (2) company-wide regional deployments, and (3) as a global service. Internal deployments (1) are usually described by instances of a Speckle server running locally behind a company firewall. They are not accessible from the outside world, nevertheless designers, engineers and project managers can rely on them for their day-to-day work and they guarantee full data ownership and safety. Company-wide deployments (2) usually run on a cloud provider in the company’s geographical region, and are open to the wider Internet. This allows for interaction with clients and external stakeholders, and they bypass issues arising from latency by their geographic co-location with the majority of their users. Finally, global services (3) are a multi-tenant scenario, in which more companies use the same server, provided and catered for by an external provider, such as a BIM consultancy.
At a larger scale, these individual deployments give birth to a decentralised model of data exchange (Figure 62). To clarify, within one server data is centralised, nevertheless, end-users may have accounts on multiple servers, from different parties or companies. Thus, they act as bridges between different, decentralised data sources. Data residency and access is decided organically, based on needs, and each stakeholder has control over what parts are exposed to outside collaborators.

Summing up, design information ownership and access can be situated and managed organically around stakeholders and Speckle deployments and, in turn, selectively opened up based on needs arising from the collaborative requirements of each individual design task. This opens up and enables new avenues for looking at the social and information architecture of collaboration in design projects—simple or complex; avenues which go beyond the current single-source-of-truth methodologies and lead towards better supporting human collaboration and cooperation.
8. Conclusion

During the course of this thesis, a series of alternative approaches to digital design data communication were assessed. Taken together, they articulate a new technical platform for communication inside the AEC industry. Starting from a theoretical analysis of communication as both a social and technical phenomenon, three specific research questions were formulated that deal with distinct aspects of the communicative and collaborative act in the design disciplines, namely (1) representation, (2) classification, and (3) transaction.

From a methodological point of view, the research was embedded within a living laboratory context and, consequently, had an important applied component. This consisted of a new programmatic framework for digital design communication, named Speckle, which allowed us to analyse and assess (1), (2), and (3) against existing collaborative methodologies from the AEC industry. Speckle was developed throughout the research project as an open source, freely-available software platform that underwent continuous validation and testing both “in the wild”, from various industry stakeholders, as well as through three directed focus group meetings.

At a theoretical level, this research project argued that existing digital information exchange methodologies from the AEC industry enforce a technical model of communication on what is an essentially psycho-social and cognitive process, therefore leading to an unproductive design environment that restricts the emergence of shared understanding. By looking at (1), (2) and (3) from an integrated perspective of both technological affordances and social communication requirements, the research questions have expanded the tree aspects into the following research directions:

1. **Data representation**: what are the advantages and limitations of a lower-level, schema-abstract, composable object model over those presented by an ontologically complete higher-level standard?

   The analyses in *Chapter 4, Data Representation*, have shown that ontological revision at a representational level—how design objects are defined—is a naturally occurring step throughout the design process. *Sections 4.1 (Composable Data Structures) and 4.2 (Encoding Existing Ontologies)*, revealed that a lower-level, composable object model can simultaneously accommodate both ad-hoc, user-driven, data structures as well as programmatically match existing, predefined higher-level object models (such as IFC, or BHoM). Following, *Section
4.4 (Managing Ontological Diversity) puts forward a methodology through which multiple object models can be supported in a digital communicative process without compromising on consistency and rigour.

The empirical analysis from Section 4.3 evidences that end-user driven ontological revision tends to minimise the complexity of the data structures involved, with a relatively large difference between the average tree depth of end-user created objects (1.97) and those coming from existing schemas (3.43). This translates in smaller object sizes (up to seven times) that, in turn, lead to a leaner digital data exchange volume.

Overall, the claim that, as an industry, AEC needs a singular and unique object model in order to consistently exchange digital design information is invalidated. Nevertheless, a major limitation is the issue of asymmetrical information codification and de-codification that occurs when communicants do not fully share the same knowledge context. This limitation, and its mitigation by enforcing one-directional information flows and object immutability, was addressed in Section 5.5 (Object Identity, Immutability & Data Deduplication).

2. **Data classification:** what are the advantages and limitations of an object-centred classification approach to digital design data as opposed to a file-centric collaboration methodology?

In Chapter 5, Data Classification, the analysis from section 5.2 (From Sharing Files to Curating Data) shows that, as opposed to a file-centric collaboration methodologies, an object-centric approach may impose ontological revision at the content level by requiring actors to negotiate what information they share and why. From the implementations and case studies presented in section 5.6 (Data History) and 5.7 (Applications), one finds that this leads to increased relevancy of data and a reduction in overall communication noise: instead of sharing information “in bulk”, any design data that is communicated must have a recipient and a direct use; furthermore the communicative network between stakeholders evolves organically.

The productivity of this curatorial approach to data communication was validated as well through the empirical analysis of data collected from Speckle. Sections 5.7.1 and 5.7.2 showed that end-users take advantage of the classification freedom offered by an object-centred approach by breaking down files in an average of two and half different groupings, and subsequently reassemble them in different “federated” models, containing an average of 2.78 different sources.
of data. In terms of actual storage size, the performance of the object-centred approach was empirically observed to be twice as high as a traditional file-based one (Section 5.7.3).

Nevertheless, the bottom-up evolution of “ad-hoc” federated models can prove to be too difficult to manage in a larger setting, and, as such, detrimental to the design process. This is because the sequentiality of information exchanges between actors, in a digital setting, is lost. As such, a key limitation is revealed as the lack of a clear communicative contract between participants.

3. **Data transaction**: what are the advantages and limitations of a centralised, file-based collaboration methodology as opposed to an object centric one with regards to enabling the key requirements of a communicative contract, namely nextness and sequentiality?

*Section 6.2* (Optimising Transactions: Differential Updates) shows that the transaction costs can be greatly reduced, in comparison with a file-centric collaboration model. Empirical observations from *Section 6.2.2* show that they can provide a fivefold improvement, whereas their theoretical upper performance limit is not bounded, as they are directly proportional to the size of the change itself, rather than that of the whole model.

This is reflected in the observations from the case studies presented in *Section 6.5*. Faster updates have the effect of allowing for the emergence of “nextness”, or immediacy, between information exchanges, which is a key quality of a communicative contract. As a result, the rate at which the representational revision identified in *Chapter 4* and the content level revision from *Chapter 5* proceed is increased, and, as such, act as a positive force for the emergence of shared understanding.

*Section 6.3*, *Assembling the Network* and 6.4 (*Information Propagation*) show that an object-centric approach can tie in data, author and source at any level of precision (from a whole model to an individual object) and thus allow for every stakeholder to be aware of what data they are depending on, from whom, and when it was last updated, thus making the sequentiality of a digital conversation tangible. The case studies from Section 6.5 show that this approach, while satisfactory for individual tasks, needs to be further studied so as to match the management and overview needs of a larger design process. Ongoing work in this direction was discussed in section 7.3 (*Digital Transactions and Communication*) of the *Discussion.*
Digital communication is a key infrastructural base on which the design process now operates. Within this research project, communication was contextualised as a transactional phenomenon, with both technical and social manifestations which reinforce each other. The contribution can be stated as an integrated technical and sociological reconceptualization of communication in the digital design process that challenges the existing status quo of the AEC industry. By marrying contemporary technical affordances with a user- and industry-centred analysis, this project demonstrates that existing assumptions around the need for centralised high-level standards and workflows discourage meaningful dialogue to happen and exclude vital stakeholders from the design process. A flexible digital communication framework, by providing an inferential context for dialogue to happen amongst design stakeholders, allows for the emergence and evolution of the way information is defined and structured, enabling the creation of shared values and meaning. Broadly speaking, the low impact of emerging technologies on the overall productivity and efficiency of the AEC industries (Barbosa et al., 2017; Charef et al., 2019; Dainty et al., 2017; Hong et al., 2019) can be attributed to a confusion regarding the understanding of its communicative processes, and their subsequent distortion through inadequate technological implementations. Nevertheless, this research project concludes that digital technologies can embrace the diversity and richness of the design process, enhance the collaborative aspects of the industry, and, moreover, open an accessible and ethical pathway towards a digitally integrated built environment.
9. Bibliography


Gamma, E., Helm, R., Johnson, R., Vlissides, J., 2015. Design Patterns: Elements Of Reusable Object Oriented Software. Pearson Education.


Appendix A: Tooling
Tooling History

Speckle A

speckle.xyz: Simple 3d object sharing

Release: November 2015

The first instance of Speckle was aiming to enable a very simple and direct, almost Instagram-like, sharing mechanism. It allowed users to instantaneously share geometry online from a Grasshopper model with the help of one single component. The resulting web link could be shared with anyone else and embedded a commenting mechanism through which others could leave textual remarks on the shared geometry.
**Speckle B**

*beta.speckle.xyz: Solution space exploration*

Code Repositories: https://github.com/didimitrie/future.speckle

Releases: February 2016, April 2016

The second instance of Speckle implemented the notion of generation and exploration of pre-computed solution spaces. It consisted of a Grasshopper plugin and a web application where users could upload, manage and view their pre-computed solution spaces.

The main point behind Beta.Speckle was to allow multiple stakeholders to collaboratively define shared meanings of value through negotiation based on a parametric model. The parametric model itself, raised to the status of a negotiation tool, would ideally incorporate all parameters (i.e., inputs) and performance measures (i.e., outputs) that have relevance to the stakeholders involved.

Some screenshots of the various exploration modes offered by the online viewer are shown below.

![Screenshot of Speckle B](image)

This specific interface allowed for simultaneously viewing two distinct instances of the solution space in a side-by-side view.
The parallel coordinates interface shown above allowed end-users to progressively filter the solution space by applying constraints to both input parameters as well as performance criteria. In the two screenshots above, an end-user initially applied a filter on the “Floor-to-Area Ratio” performance measure, restricting it between two intervals. Subsequently, the end-user applied a secondary filter on the number of office floors in the development, selecting a maximum of three. The interface would then display only the instances that would match the end-user’s requirements.
**Speckle C**

streams.speckle.xyz: real time communication & solution space pre-computation

**Code Repositories:**

Front-end: [https://github.com/didimitrie/Speckle-Streams-Frontend](https://github.com/didimitrie/Speckle-Streams-Frontend);


Speckle C, or Speckle Streams, was the first instance of Speckle that allowed for data transaction between multiple users. Originally, it was developed to support the parallelisation of solution space pre-computation. It allowed for drastically reducing the time required to compute the full parameter space by enabling end-users to use multiple instances of Grasshopper, spread across multiple computers, for this task. Furthermore, it provided an online interface through which a designer could easily define the amount of instances a parametric solution space should be divided into.

It also paved the way for the separation of the actual data transaction layer from the communicative transaction one: Speckle Streams exposed both a REST api for large data payloads as well as a WebSocket API for real-time event notifications.
This enabled end-user collaboration across multiple Grasshopper definitions, as shown in the video still above. This specific version of Speckle was, nevertheless, tied to the Grasshopper parametric modeller, and could not have been easily embedded into other applications.
Speckle D – Speckle.Works, the open source data platform for the AEC industry

Repositories: see following section.

Releases: March 2017 (v.0.0.1), September 2017 (v.0.0.3), April 2018 (v.1.0.0-beta), ongoing.

This iteration of Speckle is the foundation on which this research project was built on. Essentially, it was a complete re-write of Speckle Streams, with an emphasis on scalability, robustness, transactions and interoperability. While the previous iterations were centred around Rhinoceros and Grasshopper, this current iteration explicitly separated concerns to allow for other software, such as Revit and Dynamo to be incorporated in the communicative network. For more information on the current state of the art, please refer to its website (https://speckle.works).
Technical Overview & Code Documentation

At the time of writing, this project is alive: contributors continuously modify and improve the code. For end-user documentation, please check https://speckle.systems/docs/essentials/start. For developer documentation, please refer to https://speckle.systems/docs/developers/core-concepts.

SpeckleWorks Github Organisation: https://github.com/speckleworks

Speckle Platform Sub-Projects:

2. SpeckleCore: https://github.com/speckleworks/SpeckleCore
4. SpeckleDynamo: https://github.com/speckleworks/SpeckleDynamo
5. SpeckleBlender: https://github.com/speckleworks/SpeckleBlender
7. SpeckleView: https://github.com/speckleworks/SpeckleView
8. SpeckleThreeViewer: https://github.com/speckleworks/SpeckleViewer
10. SpeckleSpecs: https://github.com/speckleworks/SpeckleSpecs
11. And more. For the full list, please check the github organisation’s page: https://github.com/speckleworks

Code contributors (in no particular order):

Mishael Ebel Nuh, Matteo Cominetti, Matthew Swaidan, Paul Poinet, Alvaro Pickmans, Tom Svillans, Antoine Dao, Will Pearson, Luis Fraguada, Radu Gidei, Nic Burgers, Chris Morse. If any are omitted, please accept the sincere apologies of the author. For the definitive reference, refer to the contributors page of each repository (e.g., https://github.com/speckleworks/SpeckleServer/graphs/contributors).
Appendix B: Industry Exchanges

The following appendix contains:

1. Meeting Minutes and Report: (London) November 2017 Speckle meeting
2. Meeting Report: (Barcelona) February 2018 Speckle meeting
4. Feedback from C.F. Møller Architects
5. Open Speckle Survey
Data Communication in AEC / 1st Speckle Meeting

30th November, 2017

Location: The Bartlett School of Architecture, UCL, 22 Gordon Street, London WC1H 0QB
Room: 5.04 (5th floor)

Attendees

1. Al Fisher [AF], Buro Happold, Associate Director,
al.fisher@burohappold.com
2. Eduardo Pignatelli [EP], Buro Happold, Computational Architect,
eduardo.pignatelli@burohappold.com
3. Giovanni Betti [GB], HENN (Berlin), Head of Performance,
giovanni.betti@henn.com
4. Paul Poinet [PP], CITA / Innochain (CPH), Marie Curie Fellow (ESR),
paul.poinet@kadk.dk
5. Martin Tamke [MT], CITA / Innochain (CPH), Associate Professor,
martin.tamke@kadk.dk
6. Shane Burger [SB], Woods Bagot (NYC, via VC), Principal & Director of Technical Innovation, shane.burger@woodsbagot.com
7. Luis Fraguada [LF], McNeel (BCN), Software Developer, luis@mcneel.com
8. Will Pearson [WP], McNeel, Software Developer, will@mcneel.com
9. John Egan [JE], BIM Launcher, Director, john@bimlauncher.com
10. Radu Gidei [RG], Grimshaw, BIM Manager, radu.gidei@grimshaw.global
11. Fabian Scheurer [FS], Design-To-Production, Managing Partner,
Meeting Agenda

• 11.30 - 12.00 Arrivals & Coffee
• 12.00 - 12.30 Introduction Presentations: Shane Burger, Dimitrie Stefanescu
• 12.30 - 12.45 Tech Stack Presentation: Dimitrie Stefanescu, Paul Poinet
• 12.45 - 16.30 Workshop & Discussions
• 16.30 - 17.00 Concluding remarks & next steps
• 17.00 - 18.30 Closing drinks

Next Steps/Highlights

It has been agreed that recurring trimestral meetings shall be called to discuss and steer future development points, both organisational and technical.

Tentative date for next meeting: 15th of February 2018, Barcelona

The main discussion topics will be:

• Comparison and evaluation of various models regarding legal incorporation
• Comparison and evaluation of various models for community and governance
• Roadmap for extra feature implementations

Foundation/Organisation Incorporation Options

In order to carry on focused work and produce progress on various topic that have been deemed of importance to the delivery of Speckle as a working tool, the need for 3 working groups has been identified:
Working groups

- Legal: Management, governance, funding.
- Infrastructure & Community: Build and communicate the “Core”. Onboard users and power users. Foster community contributions.

It has been agreed that there is justifiable need to incorporate a legal entity that will allow for a sustainable future development, expansion, and enrichment of Speckle.

The exact legal nature & geographical location thereof will be discussed and agreed upon at the following meeting on the basis of participant’s provided options which will be researched by each interested party.

We are looking into creating a common communication platform for anyone to get involved in the group. In the meantime, to participate in further discussion please join the Speckle Slack “workgroups” channel. You can get an invite automatically here: https://slacker.speckle.works

Technical Summary

The following have “emerged” throughout the discussion:

- Schema-agnostic: Speckle is a meta-standard allowing for the simultaneous articulation of multiple object models. Existing standards are welcome, as well as “ad-hoc” creation of such. IFC can (should) be implemented within Speckle.
- Project Information Modelling & Process Management: Speckle can capture the human/informational dependency graph of the design process. This can be used in a both prescriptive (observer & analyse) and descriptive way (plan ahead) to track outdated “dependencies”, informational bottlenecks, decisions and communication flows.
- Soft & Hard Data: It is recognised that much of the progress of a project consists of individual interactions. In this framework it is relevant to establish mechanisms to enable the exchange of not only objective “hard” data (i.e. geometry and property changes) but also of “soft” data (i.e. conversations, references etc.)
- Versioning: optionality, soft data, decision capture are essential and can be potentially incorporated into a git like ux. Design git (api) & github.com (ui) for AEC. Extending history to objects is probably needed.
- Permissions: Managing complex project teams, distributing information across stakeholders, navigating multiple projects, all of this requires the
integration of a stable and robust systems of permissions.

- Core, Skins, and Not Core: There is a set of Minimal Viable UIs & plugins that Speckle needs to provide. Nevertheless, other flavours and derivative products can be developed organically within the community, or privately within companies.

- Reminder for more technical documentation on speckle:
  - Github organisation: https://github.com/speckleworks
  - API Docs: https://speckleworks.github.io/SpeckleOpenApi/
  - More tutorials and write-ups: https://speckle.works/doc/

Meeting Minutes

Introduction Session


Conversations with more than a dozen companies (Gensler, F+P, etc) about interop and data flows, computation engines. Collaboration towards a core platform? Interest: cooperating towards a common AEC “operating system”.

Xavier De Kestelier: Can you elaborate on Flux business model?

SB: Pivoting, greater focus on selling rather than development & transforming the industry. Doubts re long term.

Al Fisher: Echo Flux concerns: pivoting and pirouetting. How do we define the interface between proprietary IP and common infrastructure? Transformative change: coalition led by individuals like this that can create an environment which generates buy-in.

XDK: Agrees with AF: Data transfer platform needs to be stable, and not in the hands of “one” company.

Giovanni Betti: Common language of our industry - agree on the syntax. No point in developing many internal tools.

Radu Gidei: Echo Shane - trying to build common tools. Flux looking into selling ads.

Dimitrie Stefanescu: Presentation on “what speckle is” will be followed by workshops.

GB: Workshop: user cases. What is the core, what is not core. Long term sustainability of the development effort.
**DS + PP:** Presentation 1: Speckle vision (slides 1-39) was directly followed by Presentation 2: Tech Overview (slides 40-95).

*Technical Overview*

- Flat and structured data is supported:
  - Flat object collections
  - Structured objects, schema-agnostic
  - Object extensions
- Versioning & Diffing:
  - Pattern 1: option collections
  - Pattern 2: git-like, forks, commits.
  - PRs & merge operations possible
- Object Extensibility, Conversion & Translation:
  - Base: transparent JSON
  - Smart “business” objects, custom schemas, etc.
- Federated Architecture:
  - Flexible deployments
  - Internal use only, or open (via www).
  - Stakeholders have cross-domain access - constellations of cross-server collaborations
- Process Management through client tracking
  - Decision tracing, informational bottlenecks
  - Descriptive & Prescriptive
  - “Project Information Modelling”, not “BIM” (Eduardo Pignatelli)
- Client Extensions:
  - Automation
  - Custom behaviour: analysis, reporting, etc.

*First Workshop Session*

**DS:** Initiates a roundtable discussion on what would be every participant’s “dream” core features, grouped by need, ROI.

**AF:** Allow for any standards that exist to be persistent in the Speckle ecosystem. The Speckle Object contains other objects, pertaining to different schemas/standards.

**AF:** Capture the historic actions of design iterations: git-like push/pull. Record transitions and potentially identify the knock-on effect of decisions.
Martin Tamke: How do we deal with existing buildings? Or existing situations? Or different types of model?

AF: There is no “single model” - we always have multiple different representations of the same design. Identify the relationships between.

Hanno S: A little bit like a dependency graph.

Eduardo Pignatelli: We don’t want “building” information modelling, we want “project” information modelling. Store our choices and relationships.

XDK: It’s really important to get that dependency graph: who works on the project, who’s internal, who is external. See the history of this: how does the project grows, how are they managed? Analytics on this.

XDK: Useful also for team on-boarding on projects: what is the workflow of a design project?

AF: The common data is a means to an end - moving away from top down, and moving towards a self-generating diagram that can also be used as a project management tool.

Radu Gidei: Like git visualisations.

XDK: Really important UI: let’s not end up with another pallet of spaghetti. How do you collapse, simplify?

HS: Git is a good analogy. Github is built on top of git. There will be no “one” user interface. There will be more flavours, but the API will stay the same.

John Egan: Preach discipline! Team management interface - piping data from A to B, ....

GB: Hard data vs. Soft data. What about soft data? How can a simple conversation around an issue be associated

Sam Wilkinson: This can happen per geometric object/element or per stream.

GB: Get comments (+named views) back in! Slack-Speckle integration. A social media dimension!

HS: Soft data leads to changing hard data: in transaction based systems (git), you capture this in commit messages.

XDK: Everyone forgets why these decisions are made - “why is this there?”.

AF: Comments/etc. Should exist on every level.

GB: It’s a question how you visualise these things.
**Luis Fraguada:** to what extent are you using BCF?

**DS:** Open up the api in such a way that you can have various integrations: slack, mattermost, trello.

**Will Pearson:** Slack at McNeel - brings everyone together. Plenty of tools for accountability and decision tracing in programming (not necessarily linked), but no such thing for AEC.

**FS:** Industry: projects run a long time, teams are heterogenous. Everybody’s sending BCF files by email - still no proper issue management across the industry?

**FS:** IFC was started 25 years ago [probably in a meeting just like this one - RG] - but there were only software engineers, no one actually doing the job.

**FS:** I’m looking for something that I don’t need to handle emails anymore and something that I don’t need to handle files anymore. 99% of the industry wants to just press a button => discussions on the ui.

**DS:** Some of these interfaces will need to be part of the core.

**AF:** Minimal viable interface.

**GB:** People need something that they just download and get going.

**RG:** The flux ui is a module on top of the flux api.

**DS:** Speckle is the same. All ui’s are apps built on top of the speckle api.

**HS:** Dog-fooding your own api calls.

**DS:** Whatever has been done so far was a dialogue between api expansions and ui functionalities. Every single Speckle plugin is built on top of the API, as a separate “independent” module.

**HS:** A rights management per object & attribute level. This will drive complexity up: groups, permissions, etc.

**JE:** We should decouple these things into their own components - Permissions should be handled by an API gateway with rule based access.

**RG:** We need to be able to send large breps & meshes. Lift the 16mb object limit.

**DS:** I agree. These are known technical problems, which most probably can be solved easily.

**MT:** These round here are to test and evaluate - the question is can this be done in principle?
MT: Question regarding collaboration: concurrency, diffing, merging, conflicts.

FS: Versioning on two levels: on object level, and on stream level.

HS: Diff & merge should be part of the core.

DS: Cloning is there. Diffing is there.

RG: Could you outsource this to git itself?

AF: Pull two versions down into a viewer (Rhino, web, etc.) and overlay to determine how to merge.

DS: Agrees with AF. Merging is still a human operation at a higher conceptual level [in aec].

HS: Some merging could happen automatically, some not.

JE: buildingSMART are looking for a web platform to support IFC. IFC is a flawed format, but it does have defined objects, and they have the relationships between the objects. Should Speckle aim to conform to some of these kind of objects types? Building Smart is looking for something like Speckle.

AF: We do need this to work out of the box with IFC and everything.

LF: Shouldn't it be Building Smart who writes the IFC schema on top of Speckle?

FS: I would strongly support some connection to IFC, but I need an object that can go beyond IFC

RG: Are these converters pluggable? If IFC doesn't work, then try JSON, and so on.

DS: Yes, the converter architecture is flexible, and you can definitely build on top and or extend.

DS: Speckle should be like installing wordpress. A good number of people can do it, less people can develop for it.

Second Workshop Session

DS: Summary of previous session: Transaction based-systems, git for aec. Schema-agnostic. IFC as a subset. Functionality: core, wrapped with various skins (UIs). 16mb limit #SAD. Permissions, etc.: the need to define the problem a bit better. Soft data, is it embedded in the git-like system? How do you expose complex api functionality to 95%? > Seamless ux, enable intelligent workflows without the technical complexity associated. How do you allow schema definitions to happen in the ui? Process management of design? Transcend & minimise complexity levels, applicable at all stages of design.
**XDK:** How would you formalise this? Through a company? Foundation? What is it? Non profit? Crowdsourcing will not work.

**GB:** Crowdsourcing can be an interim solution

**XDK:** You will still need to structure it, almost like a unit. X amount of people working in one specific location.

**AF:** The scope and structure of what speckle is: there is a responsibility/possibility towards participating in parts of that. A structure of individuals / not being naive / formalise the community / etc is important

**LF:** Having people typing on this costs money. Economic aspect of this should not be neglected.

**AF:** Funding mechanisms and formal shaping are going hand in hand.

**GB:** It should be a legal entity that gets funded. Something that can attract public money as well as private, and contributions in kind.

**DS:** Integrate in the roadmap the current innochain PhDs.

**SW:** UCL support? They do have an enterprise fund.

**Sean Hanna:** Will look into UCL Business; something that can receive both public and private money. Actual financial support from academia.

**DS:** Academic partnerships: answer relevant and interesting questions that carry a high risk: industry is not directly interested in, or would not demonstrate a direct ROI nevertheless still

**RG:** Precedents with Nodejs and Mongodb: open source cores governed by a foundation.

**LF:** Other efforts like Mozilla: had to create legal entities to fund all the other stuff they are doing.

**MT:** What about the balance between the various parts, private or not, of the ecosystem?

**HS:** Specific things do not need to be open source. The question is how the Core is financed.

**AF:** Agrees with HS. The architecture of the core needs to be very clear so that this private-public separation is meaningful and clear.

**JE:** What about charging membership fees? The board members get to influence the development.
LF: Stakeholders can open design lines depending on fee. These are all models that should be compared...

XDK: We need to get an overview of what kind of models are out there. Who would do that?

DS: We should all look into this?

GB: I have a shortlist! We should all put together our shortlists and go through them.

XDK: Location? Post-brexit UK?

JE: It probably should be London? Or Ireland. Good tax system.

RG: Vehicle to take these conversations further: setting up a repo for discussion around these issues.

WP: And even the project management system on github is good, it has improved.

JE: How regularly will we have these gatherings? Quarterly?

GB: Agrees. More often it doesn’t make sense, twice a year may be to little.

XDK: Next time: discuss scope and legal entities. Compare options and define roadmap.

JE: Setup several workgroups, report and discuss them next time.


XDK: Start talking with our money people inside each company.

DS: We should also look into expanding the current group of people.

GB: Yes, and we should use our own discretion and common sense in expanding the group. Expansion also into other parts of the industry, ie construction.

SW: Fabricators too.

DS: Discussion on the actual empowerment of SMBs to access and use these technologies. Wordpress is a good model to aspire to: it can be deployed with minimal technical knowledge, and with more expert knowledge it powers 30% of the internet reliably.

[session is called]
Data Communication in AEC / 2\textsuperscript{nd} Speckle Meeting

15th of February 2018

**Location:** McNeel Europe, Carrer de Roger de Flor, 32, 08018 Barcelona

**Attendees:**

1. Al Fisher [AF], Buro Happold, Associate Director, al.fisher@burohappold.com
2. Giovanni Betti [GB], HENN (Berlin), Head of Performance, giovanni.betti@henn.com
3. Paul Poinet [PP], CITA / Innochain (CPH), Marie Curie Fellow (ESR), paul.poinet@kadk.dk
4. Luis Fraguada [LF], McNeel (BCN), Software Developer, luis@mcneel.com
5. Will Pearson [WP], McNeel, Software Developer, will@mcneel.com
6. Hanno Stehling [HS], Design-To-Production, Head of Development, stehling@designtoproduction.com
7. Matteo Cominetti [MC], Arup, Automation Engineer, matteo.cominetti@arup.com
8. Max Thumfart [MT], Konstru, Founder, max@konstru.com
9. Mark Pittman (ODS Engineering), Founder, mark.pittman@ods-engineering.com
10. Jon Mirtschin (GeometryGym), Founder, jon@geometrygym.com
11. Dimitrie Stefanescu [DS], Bartlett UCL / Innochain, Marie Curie Fellow (ESR), d.stefanescu@ucl.ac.uk
Meeting Agenda

• 11.00 - 11.30 Arrivals & Coffee
• 11.30 - 13.00 Speckle Technical Updates & Discussion
  1. The Rhino Plugin: audience, requirements, how it works.
  2. Speckle, Schemas & Standards: How to normalise the approach, ie make it safe. Benefits & Freedoms; Other language environments?
  3. Server updates: scalability, deployment
  4. Integration with existing partner efforts - de-duplicating future dev work and accelerating feature deployment

• 14.30 - 15.00 Foundation & Organisation
  1. Existing models: advantages, disadvantages; Mozilla, WordPress, etc.
  2. Structural Funding Opportunities: Google Summer of Code, UCLB, InnovateUK, Innovation Fund Denmark, H2020 Grants, Eurostars
  3. Discussion: other opportunities, academic involvement & current innochain partner involvement (can be merged with point 2.d - integration with existing efforts)
  4. Use Case Workshop: Develop a list of use cases for Speckle.

• 15.00 - 16.00 Next Steps
  1. What opportunities do we go for?
  2. Who writes the grants/applications, and by when?
  3. Deadlines & overall timeline for foundation setup and organisation.

Meeting Summary

Technical update:

• Server is now fully stateless, more scalable. Redis now a dependency.
• Schema agnostic objects & serialisation (see https://speckle.works/blog/schemasandstandards/)
• Demo of the Rhino Plugin (soon will be online)

Discussion

MVP: What should be the minimal viable next release for Speckle?

• Rhino - Blocks Support
• Basic Auth, permissions, team + admin ui
• Beyond Jan 2019 > Develop as many clients as possible; GSA, Tekla, Revit / Dynamo, Etabs, Excel, other integrations
**Timeline**

Dimitrie will ramp down development efforts starting no sooner than a month from now, and for sure April 24th going forward in order to be able to write up. Will continue to be involved, but much less on actual production. A basic preliminary timeline:

- Up to September 2018/January 2019: find, document, establish and initiate incorporation of Speckle, access and secure funding for development
- After September 2018/January 2019: start developing new client plugins and ramp up general programming effort, potential service offerings as well as investigate and develop more complex ui/ux patterns (ie, versioning).

**Alliance**: Mark Pitman presented a vision on how a group of smaller companies (ODS Engineering, BIM Launcher) working in the same field could potentially team up and access various EU funds (Eurostars). Main bottleneck from acting on this is, at the moment, a lack of a clearly defined common vision and integration roadmap.

**Speckle Incorporation**: Potential way forward: Non-Profit foundation coupled with a For Profit entity. The Foundation would govern the core and mediate development interests, while the For-Profit entity would allow for absorption of higher amounts of capital and offer service based solutions.

**Action Points:**

- Develop a business plan for both non-profit and the for-profit entity.
- Establish a common vision
- Letters of support from the industry partners
- Above should materialise in the form of logos, etc. on the website
Data Communication in AEC / 3rd Speckle Meeting

26th April, 2018

Location: The Bartlett School of Architecture, UCL, 22 Gordon Street, London WC1H 0QB
Room: G01

Attendees:

1. Al Fisher [AF], Buro Happold, Associate Director, al.fisher@burohappold.com
2. Eduardo Pignatelli [EP], Buro Happold, Computational Architect, eduardo.pignatelli@burohappold.com
3. Giovanni Betti [GB], HENN (Berlin), Head of Performance, giovanni.betti@henn.com
4. Paul Poinet [PP], CITA / Innochain (CPH), Marie Curie Fellow (ESR), paul.poinet@kadk.dk
5. Luis Fraguada [LF], McNeel (BCN), Software Developer, luis@mcneel.com
6. Fabian Scheurer [FS], Design-To-Production, Managing Partner, scheurer@designtoproduction.com
7. Marios Tsiliakos [SW], Foster + Partners, Computational Designer, mtsiliakos@fosterandpartners.com
8. Matteo Cominetti [MC], Arup, Automation Engineer, matteo.cominetti@arup.com
9. Xavier De Kestelier [XDK], Hassel, Principal and Head of Design Technology and Innovation, xdekestelier@hasselstudio.com
10. Robert Aish [RA], The Bartlett, robert.aish@ucl.ac.uk
11. Ralf Lindemann [RL], Pilbrow And Partners, Head of Digital Development, RLindemann@pilbrowandpartners.com
12. Dimitrie Stefanescu [DS], Bartlett UCL / Innochain, Marie Curie Fellow (ESR), d.stefanescu@ucl.ac.uk

Meeting Agenda

The main goal of this third speckle meeting is to get actionable points on the issue of creating a sustainable base for developing this project further. Therefore, the technical aspects are kept short, and we would ideally have a longer second session discussing incorporation logistics.

NOTE: Are you across the Atlantic or going to SmartGeometry in Toronto? Can’t make it to London? Then join us on the 10th of May for dinner in Toronto (more info
on the day).

13.00 - 13.30 Arrivals, Meet & Greet
Roundtable introductions (who, where, why).

13.30 - 14.15 Tech Update
15 mins presentation: Speckle 1.0.0 fixes, novelties, directions & hooks for future functionality + 30 mins discussion. Suggested reading: speckle server & api 1.0.0 release

14.15 - 14.30 Coffee Break
They make good coffee downstairs.

14.30 - 16.00 Founding The Foundation [NGO, LLC, CIC]
10 minute presentations: Luis Fraguada, Mark Cichy, Giovanni Betti. Suggested reading: The Economic Case For Open Source Foundations

Discussion points:
Legal Entity Structure (note: the question of Speckle's FOSS nature is not under discussion)

Role & Mission

Partner Contributions & Benefits

• What defines a Partner?
• What defines a Contributor?
• What are the associated benefits for each?
• Minimums for participation under definitive categories?
• What are the categories of contributor?

Structure & Operations:

• Board Members
• Reporting and deliberation structure?
• Attribution & Marketing: Marketing leverage linked to contributions?

16.00 - 16.30 Summary & Conclusions

16.30 - 23.00 Pub Session Nearby
Feedback from C.F. Møller Architects

[Author’s Note: This lengthy review of Speckle is based on its pre-1.0 release. It is included here as a reference and as an insight into the kind of conversations that were ongoing throughout the duration of this project. They are reproduced here with the permission of the author, Ejnar Brendsdal.]

This is a summary of comments regarding a presentation of internal tools at C.F. Møller on the 29th of June 2017. It concentrates on notes and discussions around the open source collaboration tools developed under the name of Speckle by Dimitrie Stefanescu as part of his PHD project. This summary is fully a subjective point from the perspective of the author and does not represent an official opinion of the company C.F. Møller.

The presentation demonstrated tools and workflows developed in different applications including Rhino + Grasshopper and Revit + Dynamo. A part of the demonstration also showed an example of the possibilities of online collaboration through Speckle. The 3D model used in the demonstration was uploaded in different layers to the speckle server and was received at another machine in the same office. The model was also evaluated in the web browser of a third machine and shown on a mobile phone to the people present at the presentation.

Overall reception from the participants

The overall impression from the participants of the demonstration was very positive. The demonstrated capabilities of the tools were in everybody’s opinion very impressive and the integration with the Rhino through grasshopper was well done. This even though it is clear that the tools are still in early development. There was much interest in the specific features of the tools and the possibilities they will open in the pipeline of the office. Also a sharp focus was placed on the stability of the connection of the tools and the speed of updating geometry and data.

Feedback from participants

A few questions and points came up during the demonstration. These were answered to the best of my knowledge. A short summary of each follow:

• Speed:

The speed of updating models and data was noted. The participants were impressed with it, but it was pointed out that this particular demonstration was running on the speckle demo server and that the used geometry was rather simple.

There was a question if connecting big models to the server might be slower and I explained that while that was true, the speed of updating the models would properly
not affect the workflow as much as the rendering of the models in the browser. An update interval of several minutes would not be noticeable when everybody is sitting concentrating on their own part of the model anyway. It was noted that some kind of caching or selective uploading of only dirty data could be used. I don’t know if that is already the case. Overall the responsiveness of the connection was very well received. Well done.

**UI of the web interface.**

It was pointed out that the controls of the web viewer was unintuitive. I commented that this was only due to the early state of the tools. Some feature requests mentioned by the participants:

Everyone had the opinion that a set of icons for zooming and especially “zoom extends” was needed. Also, a way of setting an initial view when opening the scene for the first time would be practical. It was the opinion that the touch controls worked differently from most other mobile 3D viewers. This would be one finger drag = rotate, two finger drag = pan, pinch = zoom. Whether this is indeed an *de facto* standard of the web I don’t know. None could point to other apps than Google Maps using it.

**Reviewing**

A way of interacting with the geometry in the server was expressed as a wish. After a small discussion it was the consensus that the full power of editing geometry in the browser/mobile was way outside the scope of these tools. Instead a way to manipulate data associated with the geometry was better. This data could in turn affect the appearance of the models through the parameters in the modelling environment. I commented that there had indeed been an earlier version where it was possible to expose parameters from ie. grasshopper in the web interface and adjust geometry that way. This was of great interest.

- Possible plugins

There was an interest in the possible applications to use speckle with. I commented that this is worked on but will probably pick up speed after the core features of the platform is done. It was commented that an open API, SDK or just good demo tutorials for this could be created and I mentioned that as the project is open source this is only a matter of time and developer participation.

- Access to model from links

It was noted that a way to send the model or data to a recipient (ie. through a link to a browser) with a limited set of permissions would be useful. I pointed out that this
already is possible by simply only sharing the parts you want public without parameters. There might still be a point of thinking a bit about this.

- Export of models or subsets

In relation to the previous point it would be interesting if clients could download models in a common exchange format directly from a browser. For example they receive the link. Open the model on the browser. Hide the layers they don't need. Download only what they need. I noted that it actually is quite hard to find a common exchange format that meets all applications needs. But I agree it would be interesting. Maybe as simple OBJ if nothing more advanced like FBX could be made to work.

- Objects in viewer as links.

It could be useful to be able to attach behaviours to objects in the receiver. Like web events. This could make it possible for clients to click on an object in the web browser and it would open a drawing of the plan of that object or show an image or likewise.

- Alternative data representations CSV, Diagrams etc

A question was that it would also be great if more than just geometry could be uploaded. Say for instance CSV data as sheets, descriptions or diagrams etc. I don't know where/if this would go in terms of future plans for speckle.

**Overall experience from the demonstrator**

It is my opinion that the speckle tools in their current iteration are headed in a solid direction. The demonstration was easy to set up aside from a minor hiccup with downloading an old release. From clean install to running demo took 5 min. The documentation even at this early stage is clear albeit in the form of blog posts. A single place to gather all tutorials might be good for the future. The support through slack directly from Dimitrie has been phenomenal with help within minutes. This might not always be the case but again, being an open source project the communities and support tend to grow with the tools.

**Notes from demonstrator**

I am very impressed with the work done by Dimitrie and other developers on these tools. I feel Rhino and Grasshopper was the right place to start and I am confident in the future of the tools in no small part because contacts inside McNeel (creators of Rhino) has shown great involvement. I look forward to see which other platforms pick these tools up. That the tools run under an open license ensures that all kinds of developers can dive into it and create plugins for their applications. This is a great advantage compared to tools like Flux.io which occupy roughly the same solution.
**Final Notes**

A single note on the current Grasshopper implementation: It would be a good idea to talk to David Rutten (creator of Grasshopper) about his plans for GH 2. In earlier posts by him it has been mentioned that the ability to attach metadata to objects will be a native feature of the new Grasshopper. It’s worth investigating what this means for the Speckle tools.

A single note on applications. Many of the applications which could benefit from these tools have different overlapping feature support. I.e. Rhino works with nurbs, Revit in CSG/IFC, Web in Mesh, Blender/Max/C4D etc in Mesh.

How to make projects two-way editable when different projects support different features. Some of these concerns have been dealt with by storing different versions of geometry on the server and only working on the subset that works in the application being worked with. While this solves problems on a visible level with some limitations to editing compatibilities, a proper solution where the data being left behind or limited is clear to the user is needed. If the geometry is based on nurbs and a mesh copy is downloaded in another application, what happens if edits are done to that copy of the mesh? Is it just overwritten. Or is editing simply not possible? Or can the mesh on the server be overwritten and the nurbs then be discarded? I realize this is not part of the features supported by Speckle yet and might not be, but it is important to be aware of the issue when thinking about the eventual future of the tools.

A small side note from after the demonstration came from one of the BIM coordinators. He asked if Speckle offered any tools to upload BIM data or if there was any plans for it. I answered that while that would be very interesting, it is probably ways off yet as it currently is neither the focus of the developer nor is BIM currently a native feature of the main development platform.

Ejnar Brendsdal

Architect MAA
**Speckle Survey**

The following digital survey was sent out on the Speckle community channels in August 2018. It is not methodologically sound; it was designed by the author together with three other core contributors as a means to gather general feedback and end-user sentiment in order to inform future development. There were 31 participants in total. Because all the questions were marked as optional, some have uneven number of responses. Other questions, because they allowed for custom user input, gathered humorous suggestions.

### What is one thing that Speckle could do better?

29 responses

<table>
<thead>
<tr>
<th>Option</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation</td>
<td>17.2%</td>
</tr>
<tr>
<td>Ease of use</td>
<td>10.3%</td>
</tr>
<tr>
<td>Communication</td>
<td>13.8%</td>
</tr>
<tr>
<td>Essentials</td>
<td>10.3%</td>
</tr>
<tr>
<td>Web interfaces (ie, viewer, etc)</td>
<td>27.6%</td>
</tr>
<tr>
<td>More BUG FIXES</td>
<td>13.8%</td>
</tr>
<tr>
<td>MS Paint Plugin</td>
<td>10.3%</td>
</tr>
<tr>
<td>Projects, 3D models, and annotation</td>
<td>10.3%</td>
</tr>
</tbody>
</table>

### Which of the following Speckle plugins have you interacted with so far?

31 responses

<table>
<thead>
<tr>
<th>Plugin</th>
<th>Number of Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grasshopper</td>
<td>26 (83.9%)</td>
</tr>
<tr>
<td>Rhino</td>
<td>20 (64.5%)</td>
</tr>
<tr>
<td>Dynamo</td>
<td>24 (77.4%)</td>
</tr>
<tr>
<td>Blender</td>
<td>2 (6.5%)</td>
</tr>
<tr>
<td>Unity</td>
<td>17 (54.8%)</td>
</tr>
<tr>
<td>Online Viewer</td>
<td>15 (48.4%)</td>
</tr>
<tr>
<td>Online Admin</td>
<td>29 (60.3%)</td>
</tr>
</tbody>
</table>
Have you used speckle in production?
31 responses

Have you (or your IT department) deployed a speckle server?
30 responses

Which communication and help channels do you prefer?
31 responses

Last one, does Speckle make you happy and more productive?
31 responses
Appendix C: Impact

The living lab context, coupled with the technical instrumentation it was meant to evaluate, has proven to be extremely successful and has taken a life of its own, growing outside the main activities of InnoChain. This is partly due to the fact that Speckle (our research instrumentation) solves not only a technical issue of interoperability and data communication, but as well a political one. Namely, this has to do with the open source licensing of the project and the fact that (1) it emphatically encourages end users to own their data, and (2) it does not make them dependent on proprietary, for-profit solutions. In contrast, the current market is dominated by a small number of software vendors, the commercial aims of which do not allow them to respond in a satisfactory manner to the needs of transparency and data ownership that architects, engineers, contractors and clients have.

Ongoing Research

The most recent development is the “AEC Delta Mobility” grant, funded by InnovateUK in late 2018, with the kick-off meeting being scheduled for mid-February 2019. Within a consortium consisting of Buro Happold (former InnoChain industrial partner), 3D Repo, Rhomberg Sersa Rail UK, and UCL The Bartlett School of Construction and Project Management, the scope of this project is to normalise and specify, based on Speckle, an industry standard for design change specification that allows faster, and more agile, “delta” updates to replace file-based exchange mechanisms that are currently prevalent in the AEC industry.

Furthermore, Speckle has been used as a technological base for research in participatory urbanism at the Future Cities Laboratory at the Singapore ETH Centre. The role played by Speckle was two-fold. In the first instance, it allowed an expert user to define a subset of a design space from a parametric model. Secondly, it allowed any number of users, without a technical background, to explore said design space and express their subjective opinion in direct relation with the well-defined (mathematical) parameter values of the model itself. In a paper currently under peer review, Katja Knecht, the lead scientist of the project, concludes that “it has been shown that the approach of sharing parametric design spaces can facilitate public engagement and support the exchange between expert and laymen in urban design”. 
Industry Adoption

A number of international AEC companies have incorporated Speckle at the core of their digital transformation efforts. Amongst the most prominent we find HOK, SOM, Arup, Sasaki, BVN, Aurecon, Dialog Design, Grimshaw, and many others. Being an open source project that explicitly does not track user behaviour, downloads, etc., the above list is incomplete and is formulated based on volunteered information from end-users. In order to safeguard the delicate neutral position of Speckle within this wider context, there are plans currently led by Arup, the author’s current employer, for the establishment of an open source foundation or trust that would guarantee, but not own, the future development of the project.

Online Engagement Metrics

The project has endeavoured to reach as many potential users and stakeholders as possible, so as to have a large and rich enough context in which it could be validated. Consequently, there were many avenues of online dissemination, which are summarised below:

- The Slack group dedicated to Speckle now numbers around 800+ members;
- The Speckle organisation’s Github account has a total of more than “100 stars” across its more than 10 code repositories;
- The Speckle forum (https://discourse.speckle.works) has an excess of 400 members;
- The project website (https://speckle.works) has around 5000 unique visitors each month;
- Social media engagement over Twitter shows an excess of 30.000 views;
- Social media engagement over LinkedIn shows an excess of 20.000 video and post views;
- Social media engagement over YouTube shows an excess of 7000 views on speckle related videos.