In this paper we address the problem of knowledge management in architectural design computation practice, reflecting on our practice at Dsearch - a design computation network within White arkitekter. As a means to investigate relevant aspects of visual scripting, we introduce the notions of code, algorithm and note. We also introduce two different modes of operation within architectural practice: morphology and development - which help us distinguish the diverse knowledge types typically occurring in the structure of visual scripts. We describe two sets of tools developed by Dsearch to continuously integrate planning and documentation with design development work. The main conclusion from our practical experience of this approach is that it allows critical reflection into an efficient workflow. This constitutes a new kind of practice based and action oriented knowledge that can be curated in the form of design narratives.

Keywords: design computation, architectural practice, knowledge management, visual scripting, Grasshopper

INTRODUCTION
This paper presents a vocabulary and a set of conventions supporting knowledge management in architectural design computation practice. The conventions comprise a set of tools and routines for Rhinoceros (RH) & Grasshopper (GH), a common 3d-modelling and visual scripting environment. These are developed by Dsearch, a network of architects specialised in design computation within White arkitekter - a large architectural practice. An earlier version of this work, in particular the Dsearch Graphic Standard, have been used external to White during the Textile Hybrids workshop at Hafen City University to facilitate collaborative development between architecture and engineering students (Lienhardt and Runberger 2017).

Grasshopper, affords the designer a high degree of freedom to organise the visual aspects of the script to be more legible, without changing the underlying graph - the specific logic that executes the script. GH also allows the graph to rearrange itself. This can for instance be utilised in a powerful way by using genetic programming (Harding 2016) to evolve new versions of scripts. Dsearch has instead chosen a nim-
ble approach, deploying small, and in some cases disposable, auxiliary objects to rearrange or gather information from the script. This paper states the case for considering the layout of a visual script as a design task in its own right - vital for knowledge management. Examples of how this kind of design thinking relates to learning and communication are drawn from the practice of Dsearch.

Such knowledge often remains the internal concern of an organisation, but the purpose of this dissemination to a wider audience is to continue an established development within the discipline of architecture, relating computation to design thinking and management (Derix 2009; Hudson 2010; J. Runberger 2012; Davis 2013; J. Runberger and Magnusson 2015; Magnusson and Runberger 2017).

METHOD
The research behind this paper has been carried out through design and through practice. The authors are themselves members of Dsearch and active in the development work being described and discussed here. To make the voice of the paper clear, the arguments and experience of the authors as researchers are consistently expressed using the pronoun ‘we’. When discussing work or established conventions at White, we refer to Dsearch and its network.

Dsearch is positioned to address questions and identify potentials in ongoing projects within White, and address them through design of new design workflows or organisational changes. The authors, being part of this network and its larger context, have collected and analysed feedback from these interventions; through direct experience as practitioners complemented by interviews of individual peers and focus groups. Expressions such as “in our experience” or “we find that” are used throughout the text; such statements should be read as originating in several years of prototyping development, facing the full complexity of architectural practice (Schrage 1999; Capjon 2004).

We define this practice based approach to research as a designerly form of insider action research (Magnusson and Runberger 2017; Brannick and Coghlan 2007); seeing practice oriented and academic knowledge as a continuous spectrum - rather than different in kind. That approach aligns with the definition of design research as placing “...specific focus on the creation of new insights and knowledge through the actual design work or professional practice” (Hensel and Nilsson 2016, pXVI) or science of design, in which design strategies from everyday practice are carefully studied and used as valid sources to generate new knowledge and understanding of that practice (Cross 2006). It is also placed within the field of research by design. This field has on the one hand identified digital technology as a common theme in many practices (Hensel and Nilsson 2016), but also pointed to the importance of the projective aspects of research in architectural practice, that go beyond internal instrumentalism, addressing additional questions (Leatherbarrow 2012). This parallels a contemporary ambition to reform architectural practice as a project of knowledge production, separate from or overarching individual building project needs. One example here could be UN Studio where computational techniques are seen as one of several important platforms (Berkel and Bos 2016).

VOCABULARY
Three ubiquitous concepts are appropriated and used with specific definitions within this paper: code is defined as ‘instructional notation, determining computation’; algorithm as ‘descriptive notation, representing code agency’; and note as ‘operative notation, facilitating project development and use’. These concepts are not mutually exclusive properties of an object, but should rather be seen as different aspects of a script. They are introduced here as part of a terminology aiming to establish an integrated understanding of computation within design practice.

The notion that one single object can express itself in several ways can be exemplified in GH by describing a component - one node of the data-flow graph, the code layer underpinning the visual, algorithmic, layer in GH. As code, each component de-
terminates one stage of the data-flow, one node in a graph, turning input data into output. As algorithm, it notates this process with name, icon, and a mouseover text description. There is also more indirect information arising as a result of the process: hints of data types and structures at input and output parameters. Together with the algorithmic aspects, such information allows the designer to construct a mental representation of not only the computation performed at that specific stage, but its significance in relation to the overall design project. Grasshopper also offers components that can display visual or textual information without affecting the data-flow. This is analogous to comments in text-based programming, but the visual aspects allow for a more intricate relationship between code, algorithm and note. In Figure 1 the textual algorithmic information provided by the GH component is for instance complemented by the inclusion into a colour-coded group and the placement next to a text panel with notes relating to project development aspects.

The authors have also found it useful to distinguish between design and development processes on the one hand, and the computational and morphological processes performed by a script on the other. While the former develop sequentially over time, the latter execute momentarily according to a predefined logic. The deterministic morphological process is thus not to be confused with the open ended design process; the morphological process is rather subjected to modification through the iterations of a design process.

Dsearch has developed sets of tools and conventions for both modes of operation, supporting designers with the planning and documentation of script development. The **Morphology Set** deals with various ways to create and collate algorithmic information, aiming to represent the data-flow processing as a process generating architectural form. The **Development Set** conversely supports quality assurance in planning and documenting project issues, beyond form generation, as well as the capture of ideas and issues of relevance beyond the specific project. These modes are strongly intertwined with each other and with the underlying code; for instance when morphological aspects are linked to design decisions taken at a project meeting.

**MORPHOLOGY**

In our experience, stringent naming of parameters facilitates mental representation by linking algorithmic description to the code. Following Woodbury’s credo of *clear names* (Woodbury and Gün 2010), Dsearch has put forward the convention of *Lineages* in order to represent parameters as they are processed throughout the data flow (Figure 2). This is done by relating the name of ‘parent’ parameters to their downstream ‘offspring’. If a module output parameter is considered the downstream offspring of an input, the input name is suffixed, captioning what difference the intermediate process brought to the Lineage.

The **Lineage Set** object lists all named parameter components and moves the grasshopper window to their most upstream instance (Figure 3). This provides a descriptive overview as well as access for
deeper investigation or maintenance. The **Renamer** object facilitates reorganisation of lineages by batch-finding and replacing parameter names. Lineages paint a rich picture of how parameters are processed in the script. Some parameters have been observed to form long lineages, using other lineages for their process without losing identity; a phenomenon correlated with significance within the design process.

![Lineage Set](image3)

This enhances legibility and comprehension in that it structures the definition into levels of scale. The visual scale cues should also aid the navigation of an *abstraction gradient*, ranging from comprehensive understanding of script agency to in-depth technical detail of the code (Zboinska 2015). The script is modularised on the code level by grouping components so as to form a coherent process. By adding specific input and output parameter components the interior of a module can be reprogrammed without losing input and output connections.

Algorithmic aspects include colour coding of the group and annotation that is legible in various scales. A concise heading is written in a big font for large scale overview. Close up, input and output parameters are arranged so that their linages can be browsed in conjunction with a comprehensive textual description of the module, entered into the **Morphology Note** (Figure 5). This arrangement is complemented with small panels of annotation highlighting vital and/or unintuitive parts of the process.

![Segment A](image4)

The **Morphicle**, a portmanteau of the words morphology and chronicle, automatically creates a textual and visual summary of the script, arranged according to its modularisation (Figure 6). All Morphology Notes and headings are grouped with input and output Lineage names to form a description of each
Figure 6
Morphicle Module, and its relation to Segments and Blocks. A visual sequence is created from selected Lineage geometry and data, and displayed via the Human UI interface.

Figure 7
Graphic Standard
This object is inspired by the literate programming approach of Knuth, urging programmers to prioritise the human reader over the computer. Knuth developed a language and a set of programs he called the WEB system where any program serves as a source language for two processes. “One line of processing is called weaving the web; it produces a document that describes the program clearly and that facilitates program maintenance. The other line of processing is called tangling the web; it produces a machine-executable program“ (Knuth 1992, p101). Grasshopper, being a visual data flow based environment, is continuously ‘tangling’ the code while a script is being developed. What the Morphicle object adds is a script that ‘weaves’ various textual descriptions together, mediating a visual layout into a kind of literature.

The Dsearch Graphic Standard stipulates a colour for the group, according to functionality (Figure 7). Examples include: referencing and baking Rhino geometry, manual control or input of parameters, executing advanced script modules, etc. One primary use of this colouring is to highlight the User Interface part of the script where parameter controls are located for use by non-specialist collaborators. This is then distinguished from the back-end Development section where the actual code is found (Figure 8).

Development
The note aspect is supported by the Development Note - a pre-configured panel component for noting down issues relating to script and project development (Figure 9). The designer enters text in the syntax of “category: heading - note”. Categories include: log, todo, feature, bug, etc. Headings can ei-
ther reference parts of the script such as Lineages or modules, or they can come from the project in the form of design concepts, evaluation criteria or contractual agreements. In this way project discussions can be coupled with specific aspects of the morphology. Conversely, project issues can be raised by the designer with minimal disruption of the programming workflow. Tangentially related ideas and concerns inevitably crop up, triggered by thinking on the problem at hand. While important and in need to be noted down, they threaten to disrupt the designers train of thought. Our experience shows that the possibility to record a thought in its relevant context minimises the time needed to describe relevant background in order to remember, or explain it to a team member - thus minimising the disruption.

The Development Note Set, collects all notes from one GH script and sorts them by category and heading. This allows the designer to see for instance a list of all tasks on the todo list, or all notes regarding a specific bug. Project management, such as time planning and task allocation, is also here supported by rich, context-specific information by zooming the GH window to a specific note and review it in the context of the script. One way to tick off an item on the todo list during a review would then be to simply delete the note.

The Autolog object continuously records project development events along with a time stamp (Figure 10). This entails changes and additions to the Lineage Set, Morphology and Development Notes as well as quality assurance data, such as filenames and plugin versions. The designer can enter information by using the log category in a Development Note. Events such as key development steps of the definition functionality and the iterative exports of definitions for the use by non-specialists are recorded in this way.

A potential development for the Developer Notes and the Autolog is to integrate them with other project communication planning and documentation tools, such as mail and meeting minutes, or a task management system, such as Trello. The key consideration here is how to establish a filter for curration of this data, so that it forms a relevant description of the project history - beyond log data. Such a Chronicle would complement the Morphicle. We believe that such cross referencing of continuously collected information, can result in new insights regarding project specific issues, methodological and organisational strategies and wider disciplinary questions. Runberger proposes to format this kind of knowledge as design narratives; rich histories of project decision-making, cross referenced with morphological motivations (2012, p82-83).
DISCUSSION
On a general practice level we can state that these conventions facilitate breaking down the dichotomy between management (as in planning, task allocation and documentation) and actual design work in projects. We argue that by integrating management into the development workflow, conditions for designer agency and increased efficiency are established. This ethos is in line with contemporary software development culture, for instance expressed by the agile programming movement: “The best architectures, requirements, and designs emerge from self-organizing teams.” For the presented conventions, Dsearch makes heavy use of developments distributed throughout the Grasshopper community - especially the Metahopper and Human UI plugins developed by Heumann, Syp and Holland. What is offered back to the community is a long experience of applying and contextualising this development in a way that provides quality assurance and knowledge management within design computation practice.

The time pressures of practice risk leading to cultures where reflection is delegated to the lowly prioritised task of documentation - separated from the tasks and deliveries at hand. We would like to stress the importance of integrating this reflection on practice with the daily design work in projects. Only the practising architect that has access to project specific knowledge, can communicate this first-hand knowledge from a critical perspective - to a wider audience. This knowledge brokering is vital for individual organisational learning, as well as the development of the architectural community of practice (Wenger 1999). We recognise the necessity to carry out this kind of knowledge production as intuitively as possible, so as not to disrupt daily practice. The design narratives, sketched out above, can hopefully strike a realistic balance between learning and production - layering quality assurance driven documentation with critical reflection.

For the wider academic audience the relevance of this research lies in the conceptual framing of the content. We see a need for a vocabulary concerning visual scripting in architectural practice, beyond software specific nomenclature. Though computer science can provide technical definitions for a directed acyclic graph based data flow programming language such as Grasshopper, much of that terminology is out of reach for the design disciplines. For architectural practice, terms must be defined that relate visual programming to issues of design development, project management and strategic knowledge management. This paper presents Dsearch terminology that to a varying degree has stood the test of time. More importantly the underlying distinctions and conceptualisations have already proved to be valuable in the internal discourse at White.

CONCLUSION
The graph, defining a computational process, is a genuinely new kind of representation for architecture. Forcing the designer to express design decisions explicitly, the representation of form is no longer constrained by a final static geometric description, or tied to a historical narrative of its design process. This notion of morphological process as distinct from chronological design process, and the corresponding graph based representation, are regarded as novel additions to design methodology with great potential for further theory development.

Zboinska proposes to annotate visual scripts with a supplementary algorithm in order to boost comprehension, knowledge sharing and collaboration (2015). This is done by the use of standard GH components placed in parallel to the script. The main argument of this paper is that it is possible to expand on this idea, by relying even further on the visual aspects of Grasshopper. Dsearch allows the algorithm to be scattered and partial during development, with its constituents carefully placed in the context of the script. Only post factum is it then curated into a coherent form.

The use of notes to relate issues regarding programming, design, project management and strategic development to the script context, is a minor feature with major implications. This cross referencing
between morphology and development concerns, creates opportunities for new insights. Continuously collected along with the algorithm, this information can be synthesised as a design narrative - a new kind of practice based and action oriented architectural knowledge.

REFERENCES
van Berkel, B and Bos, C 2016, Knowledge Matters, Frame Publishers
Cross, N 2006, Designerly ways of knowing, Springer
Davis, D 2013, Modelled on software engineering: Flexible parametric models in the practice of architecture, Ph.D. Thesis, RMIT University
Harding, J 2016 ‘Evolving Parametric Models using Genetic Programming with Artificial Selection’, Proceedings of the 34th eCAADe Conference - Volume 1, University of Oulu, Oulu
Hudson, R 2010, Strategies for parametric design in architecture: an application of practice led research, Ph.D. Thesis, University of Bath
Runberger, J 2012, Architectural prototypes II: Reformations, speculations and strategies in the digital design field, Ph.D. Thesis, KTH Royal Institute of Technology
Woodbury, R and Gün, O 2010, Elements of Parametric Design, ROUTLEDGE CHAPMAN & HALL
Zboinska, MA 2015 ‘Boosting the efficiency of architectural visual scripts: A visual script structuring strategy based on a set of paradigms from programming and software engineering’, Modelling Behaviour: Design Modelling Symposium 2015, Copenhagen, pp. 479-490