Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Per-Morten Straume

Investigating Data-Oriented Design

Master's thesis in Applied Computer Science
Supervisor: Associate Professor Christopher Frantz
December 2019

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Investigating Data-Oriented Design

Per-Morten Straume

16-12-2019

Master's Thesis
Master of Science in Applied Computer Science
30 ECTS
Department of Computer Science
Norwegian University of Science and Technology,

Supervisor: Associate Professor Christopher Frantz

Preface

This master thesis was written as part of a Master's degree in Applied Computer Science at NTNU Gjøvik. The thesis work was carried out throughout 2019. As part of the thesis, I cooperated with industrial practitioners of Data-Oriented Design (DOD) in order to get an understanding of the topic. I also cooperated with the Norwegian Virtual Reality (VR) company Vixel[1] for the case study that was conducted as part of this thesis.

DOD was chosen as the topic for my thesis as I have always been interested in the area, but also frustrated at the lack of available academic literature.

This thesis is aimed at a general computer science audience, both academics, and practitioners. However, the reader is expected to have a basic understanding of the Von Neumann architecture, particularly in relation to memory and cache mechanisms.

16-12-2019

[1] https://www.vixel.no/

Acknowledgment

This document would not have been possible without considerable help from others. I want to thank my family in general for their continued support, proofreading, and discussions.

I would like to thank my supervisor, Christopher Frantz, for his excellent guidance, and belief that this topic was worth pursuing.

I would like to thank Vixel for providing an interesting case, and for providing resources to solve the case.

I want to thank all the participants who partook in the interviews and/or case validation.

Balázs Török
Marc Costa
Stefan Reinalter
Tony Albrecht
Richard Fabian
Stoyan Nikolov

The answers and feedback you gave went above and beyond my expectations, and I am glad I was allowed to publish your insights as part of the thesis.

In particular I want to thank Dale Kim both for his great mentoring, participation in the interviews, and for guidance during the case study.

Lastly, I want to thank NTNU Gjøvik for five and a half years of exciting and educational experiences. In particular I want to thank Simon McCallum and Mariusz Nowostawski for educational, insightful, and challenging content and discussions which made my time at NTNU Gjøvik a wonderful experience.

Per-Morten Straume

Abstract

This thesis investigates the topic of data-oriented design, a topic that has received little attention within the academic community, but is starting to receive popularity within the video games industry.

Through an in-depth review of academic and industrial literature I find that there is a difference in conceptualization and discussion of DOD between the industry and academic literature. The academic literature seems to see DOD largely as a set of patterns, while the industry takes a more holistic approach, viewing it as a way to solve given problems.

The understanding of DOD gained from the literature is extended through a set of in-depth interviews with industrial practitioners of DOD. With this extended understanding I arrive at the following aspects that I consider core characteristics of DOD.

1. Focus on solving your problem at hand, rather than a generic one
2. All kinds of data should be considered
3. Decisions should be made based on data
4. Focus on performance in a wide sense

Based on this gained understanding, I apply this to a case based exploration of DOD. The case focuses on streaming Building Information Modeling (BIM) models from disk into a VR application. This understanding and case solution is then validated by DOD industry practitioners with respect to the adherence to DOD principles. The practitioners agreed that I followed a DOD approach, but pointed out flaws/limitations in the execution of the approach, which led to new insights, such as the need for a stronger focus on verification and details, more generally the validation highlighted the challenge of arriving at a generalized conception of DOD. This supports the identified core characteristics of DOD (highlighted above).

In the end, DOD is a topic that is not well researched, which offers opportunities for more collaboration between the industry and academia. Recognising that DOD is a "simple concept that is hard to master", this thesis concludes by highlighting the skills necessary to learn and apply DOD in practical and curricular settings, specifically problem-centred analysis, understanding of modern hardware, understanding of tools, as well as statistical proficiency and thus provides instructive insights. Taking this work as a starting point, finally, the opportunities for future investigations into DOD are outlined.

Contents

List of Figures

List of Tables

Listings

1 Introduction

Data-oriented design (DOD) is a software development concept shrouded in mystery. While largely absent in the academic community, DOD has been discussed within the video game industry for at least a decade. With Unity's new data-oriented technology stack (DOTS)[1] data-oriented design has been thrust into a spotlight, garnering wider attention. Still, with the impressive results presented by Unity utilizing DOTS[2], and multi-factor improvements in execution time reported by practitioners of DOD[3][4] the academic community at large seems to remain ignorant of DOD. This is unfortunate as not only does it mean that computer science students aiming for a career in video games are unfamiliar with development practices, but also that other areas within computer science miss out on practices that can greatly improve performance and with that also the quality of their products.

While multiple blog posts and presentations promote understanding of and engagement with data-oriented design, there seems to be confusion regarding as to what DOD actually entails. Cache optimization is often put at the center of attention, especially within what little there is of academic literature. However, I believe that there is more to DOD than mere cache optimization.

As indicated by the title, this thesis sets out to investigate the concept of data-oriented design. The central contributions for this thesis is an in-depth understanding of DOD, achieved by the following:

- in-depth analysis of industrial and academic DOD literature
- in-depth interviews with industrial practitioners of DOD
- application of DOD to an industrial case study with a subsequent validation by industry practitioners

While the topic of data-oriented design is usually discussed within the video game industry, for reasons we will shed light into at a later stage, and discussing it entails concepts such as low-level programming and hardware details, this thesis is aimed at the computer science field in general, and the intended audience is both academics and practitioners. However, the reader is expected to have a basic understanding of the Von Neumann[5] architecture particularly in relation to memory and cache mechanisms.

1.1 Goals and Research Questions

As previously noted the goal of this thesis is to shine a light on data-oriented design. To reach this goal a set of research questions were formulated; these are presented below. The research questions are largely explorative as there is not much research done on the field.

Research Question 1

To what extent does the conception and discussion of DOD vary between industry and academia?

Research Question 2

What are the core characteristics of DOD?

Research Question 3

How does one approach software development with a DOD mindset?

1.2 Research Methods Overview

The research questions are answered using different methods.

Systematic Literature Review

Research question 1 is answered through a systematic literature review/analysis, as defined by Fink[6] and used by Okoli and Schabram[7].

> *A research literature review is a systematic, explicit, and reproducible method for identifying, evaluating, and synthesizing the existing body of completed and recorded work produced by researchers, scholars, and practitioners.* – (Arlene Fink[6, p. 6])

Semi-structured Interviews

Research question 2 is answered through semi-structured interviews with industry practitioners, and the understanding obtained from the literature research. The SAGE Encyclopedia of Qualitative Research Methods description of semi-structured interviews[8] was used as a guideline for the interviews with the practitioners.

> *The semi-structured interview is a qualitative data collection strategy in which the researcher asks informants a series of predetermined but open-ended questions.*
> – (The SAGE Encyclopedia of Qualitative Research Methods[8])

Industrial Case Study

Research question 3 is answered by applying the understanding gained from the literature research and practitioner interviews to an industrial case. The results and process of applying the DOD understanding to the industrial case is described in a process document, which was distributed to industry practitioners for feedback and validation of the approach.

The application of the DOD understanding to the industrial case is largely action research, as described by Håkansson[9]. The feedback and validation is conducted through in-depth semi-structured interviews, as previously described.

> *Action research is performed by actions to contribute to practical concerns in a problematic situation. The method improves the way people address issues and solves problems, as well as, strategies, practices, and knowledge of environments. It is a systematic, cyclic method of planning, taking action, observing, evaluating and critical reflection. Action research often studies communities or settings with restricted data sets and, hence, qualitative methods are most suitable.*
>
> – (Anne Håkansson[9, p. 7])

Further details on the individual research methods are found under the respective chapters.

2 Structure, Interpretative Nature & Writing Style

2.1 Structure

This thesis is split into six main parts. The first part, Chapter 3, briefly introduces Entity Component Systems (ECS) and Structure of Arrays (SoA) versus Array of Structures (AoS), as being aware of these concepts are helpful before discussing the topic of data-oriented design. Following that is a literature review, Chapter 4, and interviews with various industry practitioners, Chapter 5. The literature review answers the first research question, while the interviews combined with the literature review answers the second research question. Chapter 6 presents an understanding of how to approach software development with a DOD mindset as an initial answer to the third research question. This understanding is applied to a case in Chapter 7 to Chapter 9. In Chapter 10 the application of the DOD mindset to the case and the understanding of how to apply DOD is discussed with industry practitioners, resulting in a more conclusive answer to the third research question. Chapter 11 summarises the findings, limitations, and insights gained from the thesis as a whole, before presenting possible future work.

I chose this structure because it largely follows how I executed my project. Additionally, it is how I often approach learning a new topic. I.e., by reading about said topic (literature review in Chapter 4), asking people more knowledgeable than me to explain their understanding (interviews with practitioners in Chapter 5), before attempting to apply my understanding (application of understanding to industrial case in Chapter 7 to Chapter 9). Effectively, this structure attempts to triangulate an understanding of the chosen topic.

The structure is not a traditional IMRAD structure. However, I would argue that I cover the same aspects as what an IMRAD structure does, as both introductions, methods, results, analysis and discussion points are covered. For example, questions regarding weaknesses in particular steps of a methodology are discussed in the context of describing that step. Similarly, in the literature, interview, and validation chapters, the results are commented on and to a certain extent discussed as they are presented. This is done both to respond to the questions a reader might have at the point they may arise, and to lay the foundation for future discussions. An issue with this writing method is that it leads to some duplication, but I deem that an acceptable trade-off. On a humorous note, just like current generation CPUs the author has a fondness for spatial locality.

2.2 Interpretative Nature

Throughout this thesis I am approaching the various literature and interview participants with the intent of understanding their thinking and reasoning. This means that the thesis has a strong interpretative nature, and at times speculative character (with appropriate indication). With this more qualitative nature comes a larger risk of interpretative errors. To account for this I have attempted to be transparent with my approach and provided the basis for my reasoning as far as possible. While this whole thesis is in fact my interpretation of various works, I am explicitly specifying when I am paraphrasing and commenting on works, and when I am interpreting and conjecturing based on information within that work. The intention of this writing is in no way to put words in any of the author's or participant's mouths. Additionally, interviews and the text were often analyzed together with my supervisor to avoid too wild interpretations.

2.3 Writing Style

2.3.1 First Person Point of View

This thesis is written from a first person viewpoint, in a largely descriptive manner. The choice of writing style was made both because I find it easier to read and write, but also because I do not want the reader to process this document under the false pretense that it is not subjective. I do not want to obscure the thesis's reliance on my interpretation of available and collected information.

2.3.2 Use of quotes

Throughout this thesis I have been quite liberal with the use of quotations. This is done both for transparency reasons, to leave it to the reader to assess my subsequent interpretation, and to maintain a cohesive narrative. Quotes are used both to discuss an author/participants statements in verbatim, but also as freestanding entities that, in my view, further backs up my interpretation.

2.3.3 Categorization

Multiple times throughout this thesis I will use the terms *academia* and *industry*. Unless otherwise noted, when discussing academia's viewpoints and the academic literature the discussion only concerns the reviewed academic literature, rather than the whole of academia or all academic literature. Similarly, when discussing the industry, the employed interpretation that only pertains to the reviewed industry literature. The use of such broad language is mainly for the sake of brevity, rather than to make any generalizing statements.

3 Concepts & Patterns

This chapter briefly introduces certain concepts and patterns that are useful to be aware of before delving into this thesis, in particular the literature section.

3.1 Entity Component System

As noted by Martin[10] and Fabian[11] there are many different understandings and types of entity component systems (ECS). For the purpose of this thesis I will be working with the understanding of ECS as described by Martin. Martin uses the term Entity Systems (ES) rather than ECS, from my understanding the term ES has largely been replaced with ECS within the video game industry. The reason I am presenting this understanding of ECS is because it is, from my understanding, the one that has the largest focus on aggregation, it is the one that appears in this thesis when discussing DOD, and in my interpretation the one closest to how Unity is implementing their ECS.

Entity Component System is a prevalent architectural pattern in game development. It follows a more relational database like structure[12] as opposed to a traditional object oriented structure. Furthermore, ECSs help tackle, among other issues, the problem of game logic that spans multiple problem domains[13]. There are three main concepts in ECS, Entities, Components, and Systems.

3.1.1 Entity

An entity is a behavior-less, stateless, globally unique identifier for any element, phenomenon, or concept within a game world[10], often implemented as a numerical value[12]. From here on I use the term entity to refer to the element, phenomenon, or concept within a world, while entity id refers to the numerical value itself.

3.1.2 Component

Components are behavior-less pieces of data that provide aspects such as the position, velocity, or health of an entity, and together they implicitly define what that entity is and how it interacts with the world[10, 12].

3.1.3 System

Systems are external global actors responsible for operating on all entities which has a specified set of aspects. The systems are isolated and run continuously as if they were running on their own thread. In essence, the systems are responsible for implementing behavior for the different aspects that components represents[10].

3.1.4 ECS example

To illustrate the line of thought I present the following example of a simple car racing game. A car is an entity with a unique entity identifier. A car has several components, a position component and velocity component for indicating its position in the world and its speed. It also has a mesh component indicating the visual appearance of the car. These components are stored separate from each other, not within some entity class, to tie these components together you need the entity identifier. This game also has two systems, a rendering system and a physics system. The rendering system is responsible for rendering all 3D objects in the game. The physics system is responsible for moving all objects in the game that have a velocity.

During the game loop the rendering system will go through all entities with position and mesh components and render the mesh at the entity's position. Similarly, the physics system goes through all entities with a position and velocity component and update the entity's position based on its velocity.

While not necessarily obvious from such a small example, a benefit of this approach is the flexibility it provides. If you want a normally static object, such as a rock, to start moving, simply give it a velocity component. If you want the cars to be controlled by either a player or an AI, create components and systems for both of those. Attach a player component to the desired car and AI components to the others. If you want the player to switch to the movable rock midway through the race, simply remove the player component from the car entity, add an AI component to it and add a player component to the new rock entity.

3.2 Array of Structures vs Structure of Arrays

Array of Structures (AoS) and Structure of Arrays (SoA) refers to the layout of properties within a structure or class (structure from now on) in the situation where you are dealing with multiple instances of the structure[14, 15].

3.2.1 Array of Structures

Array of structures is perhaps the more conventional approach to model concepts. Here each concept is modeled as an individual object containing all its relevant information. To support multiple instances of an object they are stored in some form of a container. This can be seen in Listing 3.1.

3.2.2 Structure of Arrays

Structure of arrays is an alternative approach where the properties of all objects are kept in parallel collections, as seen in Listing 3.2. Identifying an individual object is done by accessing all the

```cpp
struct Person
{
    std::string name;
    int age;
    float height;
};

std::vector<Person> people;
```

Listing 3.1: Array of Structures Code Example

```cpp
struct People
{
    std::vector<std::string> names;
    std::vector<int> ages;
    std::vector<float> heights;
};

People people;
```

Listing 3.2: Structure of Arrays Code Example

properties located at the same index in the different collections.

3.2.3 Discussion

There are different benefits to the different approaches. Homann[14] notes that it is easier to reuse or modify an AoS structure, making it more extendable. Benefits to the SoA approach is that it is often more suitable for Single Instruction Multiple Data (SIMD) utilization as the properties are ordered in a SIMD friendly manner[15]. Additionally, an SoA approach ensures that properties are only loaded from memory when they are actually needed[11].

Example

To illustrate the differences these two approaches have on loading data from memory I present the following example of calculating the average height of a set of people. The example code is seen in Listing 3.3, while the comments notes how the different structural approaches will be read into a cache line. The example has been largely simplified, but should still get the idea across.

In the AoS approach, when loading in the height of person *i*, the CPU will also load the age and name of that person, as these two variables are adjacent to the height in memory. However, in the SoA approach the CPU will load in the height of person *i*, but also the height of person $i + 1$. Effectively this means that we get the height of person $i + 1$ without needing to load in another cache line.

```cpp
// AoS
struct Person
{
    std::string name;
    int age;
    float height;
};

float CalculateAVGHeight(const std::vector<Person>& people)
{
    float total = 0.0f;

    // Assume 64 byte cache line, sizeof(Person) = 40 byte
    // Values loaded into cache line together with people[i].height when i = 0:
    //     people[0].name,
    //     people[0].age,
    //     people[0].height,
    //     people[1].name (first 24 bytes)
    for (std::size_t i = 0; i != people.size(); i++)
        total += people[i].height;

    return total / people.size();
}

// SoA
struct People
{
    std::vector<std::string> names;
    std::vector<int> ages;
    std::vector<float> heights;
};

float CalculateAVGHeight(const People& people)
{
    float total = 0.0f;

    // Assume 64 byte cache line, sizeof(float) = 4 byte
    // Values loaded into cache line together with people.heights[i] when i = 0:
    //     people.heights[0],
    //     people.heights[1],
    //      ...
    //     people.heights[14],
    //     people.heights[15]
    for (std::size_t i = 0; i != people.heights.size(); i++)
        total += people.heights[i];

    return total / people.heights.size();
}
```

Listing 3.3: AoS vs SoA Example: Calculate Average Height

4 Literature Study

4.1 Gathering Academic Literature

To find out more about data-oriented design in the academic sense I decided to conduct a literature review[6]. For search terms I used "Data-oriented Design" (in quotes), as that is the term I am familiar with for the concept, but also included "Data-oriented Programming" (in quotes) as I know from earlier that this name has been used as a synonym for data-oriented design. I searched for the terms through the following databases:

1. ACM Digital Library[1]
2. IEEE Xplore[2]
3. ProQuest[3]
4. ScienceDirect[4]
5. Scopus[5]
6. Springer Link[6]
7. Web of Science[7]

I also searched for the terms through Google Scholar[8], however, that yielded too many results (337 for "Data-oriented Design", and 414 for "Data-oriented Programming"). Many of the results were also from fields outside of computer science, for example workplace health: "Design principles for data- and change-oriented organisational analysis in workplace health promotion"[9].

Because of this I decided to further narrow down the search results on Google Scholar. I kept the two original terms, but appended each of them individually with the terms "Computer Science", and "Software Architecture", as those were the most not too limiting but relevant terms me and my supervisor came up with. After discussions with other faculty members the terms "Functional Programming", and "Stream Processing" were also added.

Since I first became aware of the term through industry literature I decided to do a "reverse indexing

[1]dl.acm.org
[2]ieeexplore.ieee.org
[3]search.proquest.com
[4]sciencedirect.com
[5]scopus.com
[6]link.springer.com
[7]webofknowledge.com
[8]scholar.google.no
[9]Bauer, G. F., Jenny, G. J., & Inauen, A. 05 2011. Design principles for data- and change-oriented organisational analysis in workplace health promotion. Health Promotion International, 27(2), 275–283. URL:https://dx.doi.org/10.1093/heapro/dar030

search" through Google Scholar. I searched for citations in the literature to the various industry blog posts and conference presentations where I first encountered the term, as well as Fabian's[16] book on the subject.

The total number of results for each query can be seen in Table 1. The searches were executed from 24th February to 3rd March, under the assumption that Google Scholar would not add any new results while I was going through the filtering process.

4.1.1 Filtering

After completing the searches I was left with 736 articles. To find the most relevant articles I ran the results through a filtering process, which can be seen in Figure 1. While this process appears to be linear it was actually iterative as I was dealing with such a large amount of papers. For example, when I ran the language filter I was still dealing with so many papers that I could not manually check them all, so I just checked the papers with titles in foreign languages. If I found articles written in foreign languages but with English titles later in the process I would simply go back and add those articles as being filtered out in the language filter.

Duplicate Filter

I first removed all duplicate entries within the results. This filtering process was based on the titles of the results. Filtering based on titles was a result of time constraints, as it would not be feasible to go through and manually check all articles for duplicates. The problem with this approach is false positives, i.e. that different articles with the same title would be removed. To avoid this I appended generically titled results with the author name or some other unique identifier.

Language Filter

After removing duplicates I filtered out all results in languages other than Norwegian and English, as these are the languages I am comfortable with at an academic level. I did attempt to translate some results that looked to be relevant through Google translate[10], however, I found several issues with the translation that made me question my interpretation of the article, and therefore decided not to pursue translations of other literature.

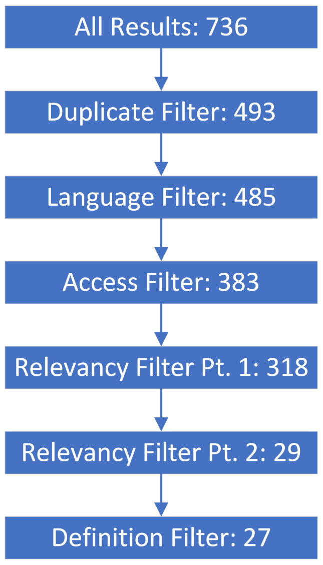


Figure 1: The filtering process for academic literature

[10] https://translate.google.com/

Search term: "Data-oriented Design"	
Database	Total Number of Results
ACM Digital Library	2
IEEE Xplore	4
ProQuest	45
ScienceDirect	20
Scopus	16
Springer Link	32
Web of Science	8
Search term: "Data-oriented Programming"	
Database	Total Number of Results
ACM Digital Library	9
IEEE Xplore	3
ProQuest	33
ScienceDirect	18
Scopus	15
Springer Link	38
Web of Science	6
Google Scholar Searches	
Search term	Total Number of Results
"Data-oriented Design" AND "Computer Science"	129
"Data-oriented Design" AND "Software Architecture"	36
"Data-oriented Design" AND "Functional Programming"	14
"Data-oriented Design" AND "Stream Processing"	3
"Data-oriented Programming" AND "Computer Science"	189
"Data-oriented Programming" AND "Software Architecture"	31
"Data-oriented Programming" AND "Functional Programming"	44
"Data-oriented Programming" AND "Stream Processing"	13
Reverse Indexing Searches	
Source	Total Number of Results
Llopis[17]	5
Llopis[18]	8
Fabian[16]	12
Acton[19]	3

Table 1: Databases and search-term outcomes

Access Filter

At this phase I went through all the articles and removed all I did not have complete and legal access to. I looked up each article on Google Scholar, and checked the alternate versions of the article in case I did not have access. If none of the alternative versions worked I would go on to the site where I originally found the article. I also removed some seemingly English results from pages in other languages (such as Chinese) that required me to register or log in. Similarly, I removed citations where I did not manage to find the original work, or did not have access to it.

Relevancy Filter pt. 1

While Google Scholar and ProQuest were the sources who brought in the most hits, they also seemed to be the ones who went the widest and therefore also brought in a lot of irrelevant results. I first removed clearly irrelevant results, such as patents, subject indices, lists of recommended books for software developers, description of PhD programs, etc. I also removed all results where the terms "data-oriented" & "data oriented" were not mentioned within the text. In certain papers I could not search for any term that contained whitespaces. In these cases I just searched for "data" and looked through all the results. In the cases where the text was not searchable; an uncommon occurrence, I quickly skimmed through the whole text if the paper in question was of reasonable length, or read through the table of contents if it was a larger document. Furthermore I removed results from other fields, such as the workplace health article mentioned earlier[11] and some cybernetics related studies.

Relevancy Filter pt. 2

At this point I was left with results from within computer science, and I started removing results that were clearly discussing a different data-oriented design/data-oriented programming than the one I was interested in. The term "data-oriented" is very broad and used to explain many different concepts, which is why a lot of results were filtered out at this stage. Table 2 shows a selection of examples of where the term data-oriented appeared, more examples can be seen in Appendix C. Based on this list it should be clear that the terms data-oriented design and data-oriented programming are very generic. Each use of the term probably warrants its own investigation, but that is outside the scope of this project. I will focus on the sources discussing the data-oriented design that I am familiar with from the video game industry.

Definition Filter

At this point I started filtering out any papers that did not contain any description or definition of data-oriented design or data-oriented programming. I searched for the terms "data-oriented" & "data oriented" in the same manner as described previously, and removed any results that did not go any further into the term. The assumption of this filter is that any description or definition actually

[11] See Section 4.1

Use of data-oriented term	Source
Data-oriented as encapsulation of data	Romanovsky, A. & Zorzo, A. 2001. A distributed coordinated atomic action scheme. *Comput. Syst. Sci. Eng.*, 16(4), 237–247
Data-oriented as exposing of data	Joshi, R. April 2007. Closer to the edge. *Electronics Systems and Software*, 5(2), 14–18. doi:10.1049/ess:20070202
Data-oriented architecture as opposed to service-oriented architecture	Vorhemus, C. & Schikuta, E. 2017. A data-oriented architecture for loosely coupled real-time information systems. In *Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '17, 472–481, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/3151759.3151770, doi:10.1145/3151759.3151770
Data-oriented design in AI as data mining	Soler, J. *Orion, A Generic Model for Data Mining: Application to Video Games*. Theses, UBO, September 2015. URL: https://tel.archives-ouvertes.fr/tel-01201960
Object oriented design as a combination of data-oriented design and operation-oriented design	Schach, S. R. *Object-Oriented and Classical Software Engineering*. McGraw-Hill Pub. Co., 8th edition
Data-oriented design as data delivery from a server to a mobile device	Ghosh, R. K. *Data Dissemination and Broadcast Disks*, 375–407. Springer Singapore, Singapore, 2017. URL: https://doi.org/10.1007/978-981-10-3941-6_12, doi:10.1007/978-981-10-3941-6_12
Data-oriented programming as an attack method in information security	Wang, Y., Li, Q., Zhang, P., Chen, Z., & Zhang, G. 2019. Dopdefender: An approach to thwarting data-oriented programming attacks based on a data-aware automaton. *Computers & Security*, 81, 94 – 106. URL: http://www.sciencedirect.com/science/article/pii/S016740481830659X, doi:https://doi.org/10.1016/j.cose.2018.11.002

Table 2: Examples of use of the term data-oriented

resides close to the use of the term.

4.1.2 Final Papers

In the end I am left with only 27 results, 8 of these results I consider to be industry literature, and those will be discussed in the industry literature section instead. Of the remaining papers only 4 are what I would consider traditional academic published papers, the rest is a collection of PhD, master, and bachelor theses, and some seemingly non-published papers and/or technical reports. In addition to these papers I added a masters thesis, and a bachelors thesis, which were found on an ad-hoc basis when I was Googling the term "Data-oriented Design". Due to the scope of this project I will not be reviewing the non-published papers and/or technical reports.

4.2 Academic Literature

In the following section I will discuss the papers that made it through the filtering process. The papers will be discussed on an author basis, grouped on their academic merit, starting with traditional published academic literature, then PhD theses, master theses, and finally bachelor theses. The papers within each group are ordered chronologically.

4.2.1 Published Papers

Derek S. Liechty

Liechty[20] looks into combining object-oriented design with data-oriented design to create an application for running direct simulation Monte Carlo (DSMC) algorithms. He notes that "*DOD shifts the perspective of programming from the objects to the data itself: the type of data, how it is laid out in memory, and how it will be read and processed during program execution.*"[20, p. 3]. Liechty highlights that following object-oriented practices leads to bad memory access patterns and cache misses, stating that while procedural programming and object-oriented programming both focuses on code, DOD focuses on the data itself. Furthermore he gives an example of a particle class, created in both an object-oriented fashion and a data-oriented fashion. In the object-oriented example the particle class simply stores the variables it needs as regular member variables. In the data-oriented case most of the variables are turned into pointers which point to contiguous arrays on a per property basis.

Comments

Interestingly Lietchy does not go into the origin of data-oriented design, nor does he seem to cite any literature on the concept.

Fontana et al

Fontana et al.[21] notes that just like games, physical design tools must handle large amounts of data with stringent performance requirements. They describe data-oriented design as an alternative to object-oriented design, stating that it is "*one of the most important concepts employed during the development of game engines...*"[21, p. 1], while also attributing its introduction to Sharp[22]. Fontana et al. also describe data-oriented design as a development concept which "... *focuses on how data will be organized in memory* ..."[21, p. 1]. They explain that in data-oriented design the properties of a concept is stored together in different arrays on a per-property basis, rather than as member variables in a class as in the traditional object-oriented model. Accessing these properties are done through the use of indices, and an instance of an object (in traditional object-oriented terms) is the collection of all properties that share the same index. They highlight that the benefit of this approach is that algorithms that only needs access to a set of properties can be fed with the arrays containing only the properties the algorithm needs, rather than a whole object containing lots of unrelated data, which improves cache utilization. A challenge with this programming model is that most programmers are not used to it. Furthermore, care must be taken to ensure that the arrays stay contiguous upon removal and addition of data. Using an entity component system pattern is suggested by Fontana et al.[21] as a way to deal with these challenges. Their explanation of an entity component system design is that an instance of an object (entity) is defined through all properties (components) that shares a unique identifier, rather than just an index. Furthermore, the entity system takes on the responsibility of creating and destroying entities, ensuring that the properties always remain contiguous. In their study they implement a Electronic Design Automation (EDA) tool using Ophidian, an engine they made for EDA development with a data-oriented design model[23]. They benchmark their solution in terms of performance against a more object-oriented solution, finding that on average the DOD solution is 94% faster for one of the test cases, while the object-oriented solution was only 6% faster in the other test case[21]. They state that the reason the DOD solution is faster in one of the two test cases is because that test fully exploit cache locality by only having one entity component system and one property. In the other test there are multiple systems and properties. These are not stored or accessed in a manner that is as cache friendly, which is why the object-oriented solution is faster in that case.

In a later paper Fontana et al.[24] goes further into the memory aspect of data-oriented design. They explain how a CPU has caches where data is stored for faster access, and how there is higher latency in the higher levels of cache (and main memory). Similarly, they highlight that the object-oriented programming model often makes inefficient use of the spatial locality property of the CPU cache. This is because member variables are stored together in memory and will be loaded into cache even if they are not used within a specific algorithm. Fontana et al. highlight that data-oriented design is an alternative here as it actually focuses on how data is organized in memory. Furthermore they point out that in the data-oriented design model one can better exploit the parallel functionality of the hardware, as data is stored in independent arrays. In the paper they investigate register clustering algorithms, again comparing a data-oriented and an object-oriented solution. They also compare parallel implementations of the two solutions. They found that the data-oriented implementation was 7.5% faster in the sequential case, and that the data-oriented implementation was 15% faster in the parallel case.

Comments

It is interesting to note that while Fontana et al. state that data-oriented design is a modern software development concept, they attribute data-oriented design to Sharp[22] in their early work[21]. However, this connection is not brought up in later works. In general Fontana et al. are quite light on where the term data-oriented design comes from, explaining the term as if it were well established, without going into its origin. Furthermore they seem to conflate data-oriented design and the entity-component system pattern, as from my understanding, the main difference between DOD and ECS in their explanation is that in an ECS you use a unique identifier rather than direct indices, and that an ECS is responsible for creating and destroying entities such that the property arrays remain contiguous.

4.2.2 PhD Theses

Joseph Carter Osborn

Osborn[25] shortly describes ECSs and notes it as a subcategory of DOD. From my understanding, he suggests that in data-oriented design (at least in terms of ECSs) one would prefer to store what would traditionally be member variables in vectors of values on a per property basis. Furthermore, he notes that ECSs leads to more predictable memory access patterns.

Comments

Osborn does not delve deep into DOD, and it is difficult to always ascertain whether or not he is discussing DOD as a whole or ECSs in particular. He cites Llopis[17][12] but does not go further into the origin of DOD.

4.2.3 Master Theses

Hazel McKendrick

McKendrick[26] states that "*Data oriented design focusses specifically on the location and efficient access of data relating to a specific problem.*"[26, p. 27]. She exemplifies this by explaining how in an object oriented approach, related variables to a concept would be stored in a single class as member variables. In the data-oriented approach a programmer would take into consideration that they are dealing with a collection of objects, and therefore adopt a SoA approach, with all the variables stored in contiguous arrays on a per member basis. McKendrick hightlights that this approach is beneficial for cache usage on the CPU side, but also that it is beneficial in relation to

[12]See page 23

GPU programming, as you can transfer only the data that you need to the GPU and access the data using thread IDs[26].

Comments

As with many of the others, McKendrick does not cite any industry literature in relation to DOD. However, it is interesting how she establishes that DOD can help in relation to GPU programming.

Mattias Karlsson

Karlsson[27] also motivates the use of DOD both as a counter to the performance gap between memory and processing speed, and pipeline stalls based on memory access. He notes that "*In Data Oriented Design the data takes the central role and a program is merely seen as a data transformer*"[27, p. 36]. While he brings up the structure of arrays approach, he also notes that DOD favours sequential access to memory, which can lead to easily parallelizable brute force solutions that can outperform more complex solutions. Furthermore Karlsson notes that "*The DOD paradigm implies that the data need to be known beforehand*"[27, p. 37] which works well for games, but not necessarily in other applications. He also makes the observation that "*Even if many aspects of DOD are well tried and tested, the paradigm is relatively new and little research in the area has been found*"[27, p. 37].

Comments

Karlsson mainly goes into data-oriented design within an appendix, unlike many others he actually draws attention to industry literature, citing Llopis[18] and suggests reading the presentation by Albrecht[3][13]. The notion that you must know the data beforehand in DOD is interesting, as it suggest that the programmer needs a certain amount of meta knowledge to make efficient use of the paradigm, and also that there might be limits to how generic solutions can be. The observation that simple brute force parallelized solutions with good cache utilization can outperform more complicated solutions with lower theoretical time complexity, combined with the predictability of DOD might help explain why the paradigm is attractive. You get simplicity combined with good performance. Furthermore, the observation that the paradigm is new and little research has been found, helps justify further looking into it.

Joel Lennartsson

Lennartsson[28] motivates the use of data-oriented design by describing the growing gap between processing speed and memory access speed. "*The main focus of DOD is to minimize the number of data transfers needed between the main memory and the cache*"[28, p. 29]. He also highlights how polymorphic behavior often lead to the use of virtual functions, where a virtual pointer table

[13]Both of which are discussed in industry literature

needs to be loaded into memory to evaluate which overloaded function to call, leading to further inefficient cache usage. Similar to McKendrick[26] and Karlsson[27] he also brings up the structure of arrays approach as a way to get better cache utilization, and also state that data-oriented design lends itself well to multi-threaded applications.

Comments

Like Karlsson, Lennartsson draws attention to industry literature, but does not seem to discuss where the term originates from. He also uses different synonyms of data-oriented design, referring to it once as data-oriented programming, which justifies the introducing the term to the literature search.

Teodor Jakobsson

Jakobsson[29] does not dive deep into DOD as a topic, but mentions it when discussing SIMD vectorization. He highlights that a structure of arrays approach better fits with SIMD vectorization, and that a structure of arrays approach is a "*... key foundation in Data Oriented Programming (DOP)*"[29, p. 38].

Comments

The main interesting aspect of this paper is the link drawn between the SoA approach and SIMD instructions.

Joakim Gebart

Gebart[30] ties together object-oriented programming and the array of structs approach, stating that "*OOP focuses on arranging data in a way that makes logical sense to the software architect and makes it easier to design programs.*"[30, p. 20]. With that view he states data-oriented design (data oriented programming in his paper) is the opposite of object-oriented programming, before moving over to state that data-oriented design is realized using a structure of arrays approach.

Comments

While some of the other papers also bring up the cache inefficiencies of object-oriented programming, Gebart is the one that has gone the furthest so far, stating that data-oriented design is the direct opposite of object-orientation. Furthermore, like most of the others, he does not seem to cite any industry literature, and does not go particularly much into the origin of the concept.

David Alexander Bond

Bond[31] brings up both ECS's and data-oriented design describing them as different programming paradigms, and leans quite heavily on industry literature, specifically Llopis[18] for the data-oriented design. He argues that "*DOD specifies that a program should be structured in a way that is* *most suitable for the processing of the data; where possible data should be grouped by type, even if it is from different objects.*"[31, p. 21]. He mentions parallelization, better cache utilization, modularity and simpler testing as benefits of DOD, while unfamiliarity, harder debugging, and interfacing DOD with OOP are considered drawbacks. For both the benefits and drawbacks he cites Llopis[18].

Comments

Bond seem to largely restate what Llopis[18] wrote in his piece[14]. However, some interesting notions can still be gained from Bond, in my view, mainly that he refers to both ECS's and DOD as different programming paradigms.

Walid Faryabi

Faryabi[32] states that "*Data-oriented design is a programming paradigm motivated by the performance gap between the processor and memory.*"[32, p. 13], with the main goal of reducing cache misses. Furthermore he lays down the foundation of DOD as following "*Construct your code around the data structures, and describe functions and methods in terms of what you want to achieve in terms of manipulation of these structures.*"[32, p. 16]. With this foundation Faryabi claims that DOD solves many problems that stem from object-oriented programming. These problems include the lack of spatial locality due to allocating objects from the heap, inefficient cache usage by fetching unneeded member variables of an object, and parallelization difficulty due to needing synchronization within objects to guard against race conditions. Similar to Fontana et al.[21] he brings up the importance of storing data in contiguous memory on a per property basis. Additionally, he also brings up the entity component system paradigm as an architectural pattern that fits with data-oriented Design. While he does not go into the origin of the concept, Faryabi states that it was particularly used during the Playstation 3 and XBox 360 era where cache misses became a performance issue.

Comments

Faryabi's thesis did not show up in the original literature search, but rather came up during the less systematic Google search for industry literature. While Faryabi gives a thorough overview over data-oriented design it is difficult to properly understand where he got his information from. He mentions different sources, but does not properly attribute which aspect he got from where.

4.2.4 Bachelor Theses

Vittorio Romeo

Romeo[33] states that "*DOD is all about the data: code has to be designed around the data and not vice-versa.*"[33, p. 35], and highlight the parallelism, modularity, and cache benefits of the paradigm. Furthermore, he also draws attention to common complaints that DOD is less convenient and safe than object-oriented programming, and that encapsulation and abstraction is exchanged for flexibil-

[14]See page 23

ity and runtime performance when following the DOD concept. These beliefs is something Romeo tries to tackle in his thesis, by implementing an ECS that makes heavy use of C++ templates to supply compile time abstractions and safety mechanisms, like multi-threading support and safe concurrent access to components belonging to an entity.

Comments

Like Faryabi, Romeo's thesis was not from the original literature search, it was included as I was aware of it before from a project that tackled similar problems as the one within his thesis. The main point of this thesis is to create an ECS, and Romeo does not seem to go that far into the DOD concept itself. Furthermore I do not always find it clear when Romeo is actually discussing data-oriented design specifically, and when he is discussing the data-oriented composition of his ECS. However, this thesis is interesting as it is the only one so far who has tried to provide a data-oriented ECS that tries to be as safe and abstraction friendly as object-oriented design.

Teo Hilthunen

Hilthunen[34] states that "*Data oriented design is fundamentally very different from the above [Object-oriented and functional programming]. Data oriented programming aims to align data based on how it is accessed in the physical memory of the computer*"[34, p. 12]. Followingly he goes into how the CPU fetches data from main memory, and the functionality of a cache as a motivation for DOD. He mainly relies on Acton[19], and retells Acton's chair example[15] arguing for the importance of understanding how data is modified. Furthermore, he notes that mixing paradigms is possible to maximize their strengths and counter their flaws, and in this context states that "*Data oriented design is an option, if the extra performance is required.*"[34, p. 13]

Comments

Similar to Romeo[33] Hilthunen states that data-oriented design can be used together with other paradigms, which has not been that common. Furthermore Hilthunen is seemingly the only one who has gone into Acton's[19] presentation. With that he highlights that it is not only memory access that is important to data-oriented design, but also to understand how data is being modified also in a logical context.

Dominique Grieshofer

Grieshofer[35] attributes the term data-oriented design to Llopis[18], and states that "*[DOD] is about arranging data optimally for the CPU to use caching to be able to transform data efficiently*"[35, p. 4]. He gives a brief explanation of how CPU's interact with the cache, citing different industry literature[16], and state that ideally processing should work on contiguous memory. Furthermore, he

[15]See page 26
[16]Discussed in the industry literature section

points out that like other optimizations, optimizing for data-locality hurts flexibility and abstractions. Like Fontana et al.[21] he brings up the problem of removing game objects from contiguous arrays, and suggest using handles and an index table to handle this problem. Furthermore, Grieshofer states that "*... the array of game objects should be sorted by their active state to get rid of branches and prevent branch prediction fails from happening.*"[35, p. 5], he also suggests that it is possible to "fold" variables into each other, like using a bitmask rather than multiple bool values, to get more information on a per byte basis. Lastly he states that "*Data-oriented design usually means that processing becomes a global scope instead of a local one, on the other hand, it also results in often simpler code.*"[35, p. 6], and that it is easier to parallelize such code. For this he cites Sharp[22].

Comments

Grieshofer[35] is the only one who attributes the term data-oriented design to the article by Llopis[18], and cites a lot of different industry literature. Like Fontana et al.[21] he cites Sharp[22]. However, I am not able to find the line of thinking that he attributes to Sharp. It is also interesting that he suggests avoiding branching by sorting based on attributes, as this implies having more in-depth knowledge about the problem you are solving at a higher level.

4.2.5 Other

Sharp

The paper by Sharp[22] is originally one that was removed during the filtering process. However, due to the attribution by Fontana et al.[21] and being mentioned by Grieshofer[35] I choose to include it. Sharp[22] argues to move away from languages that operate on Von Neumann[5] principles to better utilize the processing power, concurrency and memory in modern computers. He proposes a new language to support a data-oriented program design methodology. This methodology is exemplified by calculating payment for an employee, and follows a pattern of first defining the flow of data, before specifying the transformations to be done on the data; reaching the desired level of detailed specification through stepwise refinement. Sharp, through his example states that using their design notation, a programmer can specify a notation without placing ordering constraints on the individual operations or dataflow, allowing an implementation opportunities to parallelize the solution. The programming language suggested by Sharp is, based on my understanding, on the functional data-flow programming side. From my understanding, no implementation of this language is supplied. Apparently, Sharp goes further into this in a later paper, however, that paper is not something I have access to.

Comments

I am a bit unsure as to why Sharp gets attention in relation to data-oriented design. While the term is somewhat used within his paper, and he mentions a desire for better memory utilization and increase concurrency, Sharp does not seem to bring up the same main topics that the other academic sources do, such as the inefficiencies of object-oriented design, data organization in memory, structure of arrays, etc. However, one point that is brought up is the viewpoint that "*Any program can be regarded as specifying a system to transform a given set of inputs to a required set of outputs*"[22, p. 45]

4.3 Gathering Industry Literature

While there is a lack of academic literature on the topic of data-oriented design, a lot has been written by industry professionals. Unlike academic literature the industry literature is not stored within searchable databases, meaning that a rigorous systematic approach to gathering sources within the industry literature is infeasible given the constraints and resources of this project. The sources used in this section has been gathered from Bartolini's list of data-oriented design related resources[17] and the industry results from the academic literature search. Additionally, I have personally been interested in this area for the last two-three years, and will include any resources I am aware of that do not present themselves through the previous two methods. Many of these resources are in the form of conference presentations, blog posts, or just presentation slides. Some of these resources expresses themselves very clearly, while others are a bit more abstract. My summaries will be more influenced by my interpretation in those cases, and the intention is not to put words in any of the authors' mouths. Many of these types of resources (blog posts, magazine articles, presentation slides, etc), might look less reliable on the surface. However, from my understanding, they are the main forms for practitioners within the industry to share knowledge.

The same filtering rules apply to the industry literature as the academic literature, meaning that I require some sort of definition or description of what DOD is. If an author has provided additional works those will be included, assuming there is enough context to make the argument that they are still discussing DOD even though it is not explicitly stated.

4.4 Industry Literature

Unlike the academic literature, the literature descriptions here are not ordered by a notion of academic merit, but rather in an ad-hoc fashion.

Noel Llopis

Llopis presents his ideas through a set of articles in the game development industry focused magazine "Game Developer", blog posts, and chapters within practitioner industry literature such as "Game Engine Gems".

Llopis[18] argues that object-oriented programming and its culture are to blame for the poor per-

[17] https://github.com/dbartolini/data-oriented-design

formance resulting from bad memory access patterns. He suggests data-oriented design as a solution to these problems, stating that the difference between DOD and other approaches are that "*Data-oriented design shifts the perspective of programming from objects to the data itself: The type of the data, how it is laid out in memory, and how it will be read and processed in the game.* "[18, p. 43].

Llopis further motivate the focus on data rather than code by highlighting that programming in its essence is about transforming some input data into some output data, and that this focus on data allows a programmer to architect their whole program around an ideal data format. This format should be guided by the type of data and how it is processed, which often is homogeneous and contiguous data that allow for sequential processing. He further suggests that splitting objects into components and grouping them together by type irrespective of which object they belong to can help achieving such a format. He also argues that irrespective of the format, a programmer should try to minimize the number of transformations during runtime, and instead transform the data into the ideal format offline.

Llopis argues that object-oriented programming is too focused on singular entities, when the actual problem a game programmer is solving is having multiple entities. By contrast, Llopis states that working well with multiples is something that makes data-oriented design powerful, and draws a parallel to particle systems, which are very often implemented in a DOD manner.

> *Step back for a minute and think of the last game you worked on: How many places in the code did you have only one of something? One enemy? One vehicle? One pathfinding node? One bullet? One particle? Never! Where there's one, there are many.* – (Noel Llopis[18, p. 43])

Followingly Llopis argues the advantages of data-oriented design, bringing up better cache utilization, both in respect to the data cache and the instruction cache. He also argues that parallelization is easier due to working on smaller functions with clear inputs and outputs, requiring minimal synchronization. Furthermore, he argues that it is easier to make more intelligent high-level optimization decisions, as it is clearer how the data is transformed. Outside of optimization Llopis also argues that with data-oriented design one will end up with a more modular codebase, with lots of dependency free functions that are easy to understand, update and replace. He argues that testing becomes simpler, for example, to write a unit test one only needs to: "*Create some input data, call the transform function, and check that the output data is what we expect.*"[18, p. 44]. He also argues that when following a data-oriented design approach, one ends up with less code. "*Code that before was doing boring bookkeeping, or getter/setters on objects, or even unnecessary abstractions, all go away.*"[36, p. 254]

Llopis does however highlight that data-oriented design is different from the traditional approach programmers are used to, and that it requires practice. He also notes that it can be difficult to interface with code written in a non DOD approach[18]. Furthermore, it can be more difficult to get a big picture overview of the program due to the focus on data and small transformations. However, he notes that this is something that might be mitigated by tools or language functionality in the future[36].

In a later article Llopis[17] asks the question whether or not doing data-oriented design optimizations are worthy of its time commitment. He argues that unless a new technological breakthrough improves memory access times, the problems that data-oriented design attempts to solve will stick around for a long time. Llopis points out that replacing C/C++ would be hard, but argues that there are changes he would like to see to make it easier to work in a DOD manner. He expresses a desire for a functional language, or a version of C where functions cannot access global state or call code outside of its own scope, with defined inputs and outputs. With this he argues that one could express dependencies between the different functions, which could be fed to a scheduler that would find the best way to run the program given the constraints of the platform and dependencies.

Comments

Llopis brings up the problem of bad memory access patterns and cache utilization brought by object-orientation as a motivation for data-oriented design. However, from my understanding, the better cache utilization often seen in data-oriented design is a result from the shift in perspective to the data, rather than being the main point of data-oriented design itself, as Llopis states that the ideal data format "... *depends on the data and how it's used.*"[18, p. 43]. Similarly, from my understanding, grouping data on a type basis is done mainly because it is a helpful step on the way to discovering the ideal data format, rather than necessarily being the ideal data format.

The notion of structuring the program around an ideal data format is also quite interesting, as it implies possessing knowledge of what the data is, and how it looks. From my understanding, this information would need to be known early in the process, to avoid costly code refactoring later in a project. This line of thinking combined with the insights that most of the time a game programming problem will handle larger groups of objects, and encouragement to execute many computations offline, suggests a pragmatic approach of solving the problem one actually has, rather than a generic one.

The view of programming as data transformations is quite interesting, particularly when it is combined with the desire for a functional language. Is it just about the purity of not accessing data and functionality that has not been explicitly supplied, or would a pure functional language like Haskell[18] be ideal? My interpretation is that it would not be a pure functional language, but rather a functional-esque language which still allowed for clear mappings to the hardware and memory management. I understand this more as a desire to have clear data dependencies, rather than trying to directly avoid state of any kind. However, not being able to call code outside of a function (or its scope) is an interesting notion that I am not familiar with from other areas.

It is interesting that while Llopis[18] brings up parallelization opportunities and better cache utilization, the benefits also include other concerns, such as high-level optimizations, modularity and

[18] https://www.haskell.org/

ease of testing. These aspects have not received much attention in the academic literature.

Lastly, Llopis does not really go into the origin of the term, or the line of thinking, but does cite a blog post by Acton[37] as a resource.

Mike Acton

Acton presents his views on data-oriented design through a collection of industry conference presentations and blog posts.

He does not really go into the origin of data-oriented design, but states that the fundamentals are not new ideas. Acton states that for him, it is a response to the C++ culture, and what he calls the "*three big lies*"[19, 17:15] caused by that culture.

1. "*Software is a platform*"[19, 17:22]
2. "*Code should be designed around the model of the world*"[19, 17:30]
3. "*Code is more important than data*"[19, 17:38]

He argues that these lies leads to problems of "*poor performance, poor concurrency, poor optimizability, poor stability, and poor testability.*"[19, 25:30], and in response to these "lies" he presents a set of truths.

1. "*Hardware is the platform*"[19, 1:09:41]
2. "*Design around the data, not an idealized world*"[19, 1:09:47]
3. "*Your main responsibility is to transform data, solve that first, not the code design.*"[19, 1:09:50]

Acton presents his view of data-oriented design through a set of principles. Starting first with the line of thinking that all programs exist solely to do data transformations. In this view, data should be at the center of attention, not the code. Furthermore, he states that "*Programmer's job is NOT to write code; Programmer's job is to solve (data transformation) problems*"[19, 23:40], and that code is simply a means to this end, and with this you should only write code that adds progress towards the goal of solving the data transformation problem.

He further goes on to highlight the importance of understanding your data, and that it is key to understanding and solving your problem efficiently. From my understanding of this and his other works[37, 38, 39], data in this case is everything from statistics and patterns in the data that is to be processed, to information about the hardware the program will run on.

Solve for the most common case first, not the most generic. – (Mike Acton[19, 28:18])

With this view on data, comes a focus on coming up with non-generic solutions to given problems, with Acton stating that you can come up with better solutions when you have more context. With this in mind one should not throw away known constraints, patterns in the data, knowledge of how the data is used, or knowledge about the target platform, as taking advantage of this can lead to more optimal solutions.

An example of coming up with such solutions is when Acton discusses the Level of Detail (LOD) and Culling systems[39] in Unity's megacity demo[2]. When designing frustum culling systems for a 3D application it can be a good idea to store all the renderable objects in the scene within some sort of hierarchical structure, to avoid unnecessary bounds checks for objects clearly outside of the frustum. A possible solutions here could have been using hierarchical structures like octrees[40] or kdtrees[41]. However, rather than going straight into the discussion of which data structure to use, the team working on the culling system took a step back and analyzed the data they were going to render. They found that there already existed groupings in their data that they could use to make hierarchical judgements, and therefore putting the renderable objects into a spatial structure like an octree was unnecessary.

> *... a critical part of the whole concept of data-oriented design and ECS in particular is that data transparency, right. That you can understand, you must be able to understand, what the data looks like in order to make good decisions about your data. Your first instinct to add a bunch of moving parts like octrees or whatever your solution might have been would have been unnecessary, was in fact unnecessary in this case, because that structure already existed or something like that structure already existed that you can take advantage of.* – (Mike Acton[39, 17:52])

Another example of this is when Acton[19] highlights a node class used for the animation hierarchy in the Ogre engine[19]. He points out that the common case is not having one node, but multiple together in a hierarchy. In my interpretation, modeling a node in isolation is "optimized" for the generic case, not reality.

> *Reality is not a hack you're forced to deal with to solve your abstract, theoretical problem. Reality is the actual problem.* – (Mike Acton[19, 18:41])

Getting data can be done in several ways such as conversations with content creators, data analysis through the use of adhoc or more sophisticated tools[37], or back of the envelope calculations[19].

Because of the focus on data, he argues against the idea of world modeling in code, as seen in object-oriented design, as it implies hiding data. From my understanding, hiding data makes it much harder to understand the properties of the data, which is needed for an efficient solution. Furthermore, he argues that world modelling leads to unrelated data structures and transformations as concepts are modelled based on how they appear in the real world, as opposed to how they are stored and transformed in code, resulting in less than optimal solutions. The example he uses is different types of chairs within a video game, with chairs that are static, chairs that can be broken, and chairs that are affected by physics. From a world modeling perspective these are all chairs, and should therefore be sub-classes of chair. However, the processing and data needed for each type of chair is completely different[19].

To exemplify a data-oriented way of thinking he contrasts a "code first" dictionary lookup imple-

[19] https://github.com/OGRECave/ogre, (not able to find the exact commit he looks at, but it is mid October 2013)

mentation with a "data first" implementation. He argues that in a "code first" approach, keys and values would be stored together in memory as the programmer conceptually think they are a pair. However, in a "data first" approach the keys and values would be stored separately. This is because when we perform the most common operation (search); iterating through the dictionary looking for a key, we will only be interested in the value at the point when we find the key we are looking for. The "data first" case will better utilize the data cache, as it is not loading in values that we are not interested in[19].

From my understanding, he states that the most significant component of performance issues on "x64" architectures are L2 cache misses. He argues that these sorts of issues are not something a compiler can solve, but rather something a programmer needs to solve. For example, a C++ compiler cannot re-organize member variables based on locality of reference, or member variables being unused in a lot of cases[19].

He further suggests that organizing data based on the usage patterns and state, makes maintenance, debugging and concurrency easier[19]. He also argues for pre-computation in cases where applicable, stating that "*The best code is code that doesn't need to exist. Do it offline. Do it once. e.g. precompiled string hashes*"[19, 59:45]. From my understanding, this encouragement to preprocess is an encouragement to take advantage of knowledge about the data.

Acton also seems to elevate data-oriented design to the world outside of just performance, stating that: "*Everything is a data problem. Including usability, maintenance, debug-ability, etc. Everything.*"[19, 14:00].

Comments

I would argue that the most interesting aspect of these works are how data-oriented design is presented at a high level, with DOD at one slide being described as "*Practical philosophy not methodology.*"[42, p. 5]. From my understanding, this view on data-oriented design effectively covers the entire pipeline of creating a game. Similarly, DOD is not just about performance, but rather about solving problems efficiently for given goals, resources, and constraints. However, performance is a very important aspect within larger games, not only to ensure a smooth experience for the end user, but also to open up space for more content.

The DOD approach Acton advocates seems to be very focused on engineering and analytics. The approach is very much grounded in reality and the principles he presents, that "*different problems require different solutions.*"[19, 13:15] and "*if you have different data, you have a different problem.*"[19, 13:22]. From my understanding, he takes quite a heavy stance against approaches that get in the way of the reality of a situation, such as world modeling[19] or the idea of ignoring constants in Big O notation[43]. The programmer should make use of the knowledge they have about a problem and its context, and solve only for that. In my understanding, abstracting away hardware, or not taking advantage of knowledge of how the data is used, removes context from the solution to the problem. This not only impacts performance, but also other programmers ability to reason about what a piece of code actually does, and why it was written in a particular fashion. Similarly, trying to make a solution independent of the hardware and knowledge of the data can make the problem more difficult to solve, as a programmer potentially has to take into account edge cases that only exists within the generic context, rather than their actual context.

Furthermore, Acton talks a lot about optimization by writing for the hardware. A possible impression here could be that one should always write platform dependent code, essentially developing the same application from scratch every time it is brought to a new platform. However, this would not be feasible. The impression I get from Acton[19] is rather to acknowledge and adapt to the characteristics of the range of platforms you are working towards, such as with the example that on most x64 platforms, memory access will be a performance issue.

Richard Fabian

Fabian presents his idea of data-oriented design through a self published book. A free online beta version of the book has been available since 2013, however, this summary is written based on the 2018 first edition physical release.

Fabian[11] motivates the introduction and use of data-oriented design by the change in modern hardware. While he attributes the introduction of the term to Llopis[18], he does state that "*data-oriented design has been around for decades in one form or another...*"[11, p. 1]. He argues for the idea of everything being data (positions, images, button presses), even the instructions transforming data is also considered data. The other fundamental aspect of data-oriented design is that the transformations of data takes place on some hardware (sometimes unknown or abstract, other times clearly defined). With this, he defines data-oriented design as:

> *In essence, data-oriented design is the practice of designing software by developing transformations for well-formed data where the criteria for well-formed is guided by the target hardware and the patterns and types of transforms that need to operate on it.* – (Richard Fabian[11, p. 4])

He further establishes the principles of data-oriented design.

1. "*Data is not the problem domain*"[11, p. 4]
2. "*Data is the type, frequency, quantity, shape, and probability*"[11, p. 10]

With this first principle, Fabian explains that data-oriented design does not allow the problem domain to enter the code. The problem domain is something that can be left in the design document. This is quite different from; for example, the "compromise between machine and human" solution found in object-oriented design. He argues that while applying meaning to data (for example by naming it or contextualizing it with other data) has benefits, it also hinders a programmers ability to think of the data in its raw format (as numbers, or bytes). Extending data that has been contextualized with the problem domain, through the use of member functions, quickly leads to storing more data that might only be relevant in a subset of the cases. This tendency can often lead to classes becoming so context dependent that they cannot be reused. Fabian also highlights another issue with introducing the problem domain into the source code:

> *When you see lists and trees, arrays and maps, tables and rows, you can reason about them and their interactions and transformations. If you attempt to do the same with homes and offices, roads and commuters, coffee shops and parks, you can often get stuck in thinking about the problem domain concepts and not see the details that would provide clues to a better data representation or a different algorithmic approach.* – (Richard Fabian[11, p. 9])

He argues for a view in which "*... data is mere facts that can be interpreted in whatever way necessary to get the output data in the format it needs to be.*"[11, p. 9]. In my understanding, he argues that with this view there is a smaller chance of contextualising the data, and therefore a larger chance to avoid the issues discussed above.

In relation to the second principle, he states that data-oriented design is not all about cache misses, but rather all aspects of the data. The values and transformations are first class citizens, rather than the structure. With concentration on the data values, comes a focus on solving the problem at hand, rather than possible future problems.

> *Instead of planning to be extendable, it [DOD] plans to be simple and replaceable, and get the job done. Extendable can be added later, with the safety net of unit tests to ensure it remains working as it did while it was simple.* – (Richard Fabian[11, p. 11])

From my understanding, he later argues that while the type of some data can be generic, its values and how those are used are not. Therefore, generic or template solutions cannot exist.

> *The idea that data can be generic is a false claim that data-oriented design attempts to rectify. The transforms applied to data can be generic to some extent, but the order and selection of operations are literally the solution to the problem.* – (Richard Fabian[11, p. 15])

He suggests thinking about different aspects of the data, and how you interact with it. Such as: "*When you read data, how much of it are you using?*"[11, p. 24], "*Who does the data matter to, and what about it matters?*"[11, p. 24], and "*What information do you have that isn't in the data per-se? What is implicit?*"[11, p. 24].

Outside of the principles, he brings up that data-oriented design adapts better to changes in requirements, or input formats, as this implies some change in the data or transformation, rather than a change within the modeling of the problem domain. Data and transformations are simpler to couple or decouple based on requirements when they are separated, as opposed to being stored in a class modelled by the problem domain.

Comments

Fabian has, in my view, written the most extensive piece of work on the topic, and summarising it in its entirety is outside the scope of this thesis. Therefore this summary only covers the first chapter of the book. Fabian brings up a lot of interesting aspects further on in the book, such as suggesting a relation model for keeping data layouts extendable opening for stream based processing, discussions on a data-oriented design approach to implementing dynamic runtime polymorphism through existential processing, etc. However, from my understanding, these are rather frameworks to inspire you to think in a data-oriented way, rather than being data-oriented design themselves.

It is interesting that while he clearly attributes the term to Llopis[18] he states that it has been around for decades in different forms. The description of the book states that "*Data-oriented design is inspired by high-performance computing techniques, database design, and functional programming values*"[11, back cover]. A possible interpretation could be that these are the different forms Fabian is talking about. However, DOD is still being described as "*... a movement founded in games development*"[11, back cover], leading me to believe that such an interpretation is incorrect.

As with Acton[19] data-oriented design is described at a high level by Fabian[11], but with him describing it, from my understanding, as a methodology. Fabian's description of data-oriented design is clearly rooted in reality and engineering practice, rather than an abstract notion of how to "correctly" solve a problem. From my understanding, he also argues for non generic solutions, based on the idea that data itself cannot be generic.

The idea of data also being statistics about the values in the data and how those values are used, has been, from my understanding, implied rather strongly by Acton[19] and Llopis[18]. However, this aspect is explicitly stated as a founding principle by Fabian[11].

The foundations themselves are rather interesting, and to my knowledge quite unorthodox. In particular, the idea of not allowing the problem domain to enter the code, is one I am unfamiliar with. However, the arguments for that idea are in my view well founded, as I have experienced issues myself that were a result of thinking in the problem domain, rather than thinking about the data. My interpretation is also that the concept of simplicity in this DOD does not come from abstraction, but rather by doing only what is necessary.

In my view, performance is an important aspect in the data-oriented design Fabian describes. However, other traditional software concerns, such as testability, debugability, reusability, and extendability, are still discussed within the book in relation to DOD. Additionally, he states that DOD can coexist with other paradigms, just like object-oriented programming and functional programming[11].

Sean Middleditch

Middleditch[44] explains data-oriented design through a talk as part of DigiPen's game engine architecture club[20][21].

He lists DOD as another approach to software design. He notes that OOP focuses on interfaces and their interactions, mentioning encapsulation and abstraction as core concepts, and ease of use, maintainability and flexibility as benefits. He argues that traditional naive object-oriented ap-

[20]https://www.youtube.com/user/GameEngineArchitects/feed
[21]http://seanmiddleditch.com/digipen-game-engine-architecture-club/

proaches in games leads to unpredictable memory access patterns, and frequent and unpredictable switches between what type of logic that is executed. This approach leads to performance issues on modern CPU architectures, due to their deep pipeline architectures and caching mechanisms.

He explains data-oriented design as "*a focus on the data itself*"[44, 01:50], and how one designs data structures for maximum efficiency and easy reuse.

Furthermore, he notes that an important aspect of data-oriented design is controlling memory access patterns, with a focus on splitting data into chunks of contiguous memory, grouped based on what processing will be applied to them. Middleditch also state that "*Data-Oriented Design stresses the removal of data dependencies*"[44, 05:35]. Removing these data dependencies also means removing implied references and coupling between data, which in turn makes it easier to write efficient loops and multi-thread algorithms. Removing these dependencies can for instance be done by duplicating data across different structures. Part of the motivation for this removal is to avoid random memory access patterns to fetch needed data during processing. Which he argues helps in reducing situations where the CPU need to wait for memory.

The algorithms should be designed to work on the data needed for their specific tasks (like mass, position, and velocity for physics), but nothing else. Furthermore, there is a focus on simplistic algorithms, which are executed in a stage based manner, for example, by first completing a physics algorithm, before moving over to a graphics algorithm. Which he argues is a great way to avoid pipeline stalls, due to its predictable nature. He also brings up a focus on Plain Old Data (POD), with more primitive data types and external functions and logic closer to traditional C programming.

Middleditch further argues that traditional object-oriented approaches create very complicated architectures that are hard to reason about, due to its focus on abstractions and indirections, making it hard to actually understand what is going on. On the other hand, data-oriented design supplies simple abstractions focused on the data, removing unneeded layers of abstraction, and makes what is actually happening more clear. However, those benefits come at the cost of flexibility.

Middleditch further notes that for most hobby projects or student projects, their performance problems would not be solved by data-oriented design, but that the problems rather are a result of suboptimal use of GPU resources for instance, noting that "*Most AAA games can get by just fine without making use of DOD in core logic and game object code*"[44, 29:30]. But suggests using it in areas like physics and graphics.

Comments

Middleditch spends quite a lot of time on discussing how modern CPU's access memory and pipeline instructions, which from my understanding is a motivator to switch away from the object-oriented way of programming. Similarly, the observation that data-oriented design provides simpler code is interesting. From my understanding, this simplicity is rather in the form of seeing what computations are actually executed, rather than the world analogy simplicity that object-orientation tries to offer. While Middleditch does not specifically mention viewing programs as data transformations, that is the understanding I get from this work. He argues for algorithms with clear inputs and outputs and without data dependencies. Similarly, he draws a link to traditional C programming, where there was a clearer separation of data and functionality.

Stoyan Nikolov

Nikolov presents his interpretation of DOD through a set of industry conference presentations. From my understanding they build on the same slideset with minor variations, I have therefore only selected one of these presentations.

Nikolov[45] states that data-oriented design is a concept that has been around for a while, but was given a name and revitalized in the games industry. In his talk, he looks into applying data-oriented design principles to create a browser engine, as he and his team felt trapped by the architecture of chromium[22]. Specifically, he looks at CSS animations in web pages and compares the chromium engine against their own browser engine.

He argues that the "marriage" of data and operations found in object-oriented programming leads to issues of hidden state, which further impacts "*performance, scalability, modifiability, testability*"[45, 5:10]. With the object-oriented mindset, data and functionality that is used in very different contexts are stored together in objects, which can quickly en up as "*giant black boxes of logical entities*"[45, 5:29]. This not only impacts performance, but also increases coupling within the code. He states that while this is not necessarily the case in every code base, it is something he has seen in various code bases throughout his career, as well as different open source projects.

Data-oriented design is presented as an alternative approach. DOD is in his view rooted in acknowledging how hardware works, that memory is slow, and that cache utilization is key to performance on modern hardware. He also states that he views it as a "train of thought" and a framework for looking at code design. Nikolov[45] states that a goal of data-oriented design is to separate data from their operations. Here data is stored on a per type basis, and handed out to different systems on a by needs basis. The systems again produce new data, which is used by later systems. One should think about the transformations and contexts a data is exposed to through the entirety of this "pipeline". He uses a high level example of games, with the level definition and player actions being inputs to a system, and the pixels appearing on screen being the outputs. An important aspect that Nikolov points out is that data is only information that should be transformed by the different systems. Furthermore, data should be reorganized based on how it is used.

Nikolov argues that DOD promotes deep domain knowledge, as properly knowing your problem is required to find a good separation of data and logic. Furthermore he argues that "good" object-oriented programming (the OOP described in textbooks) has many overlaps with data-

[22]https://www.chromium.org/

oriented design, and that thinking in a data-oriented manner could also improve your object-oriented code.

In his example of a CSS animation system, he argues for designing the code around the most common used operation, being the tick/update operation. Furthermore, the common case within an animation system is that there are more than one animation, meaning that this also should be taken into consideration during the design phase. He argues that designing around the most common use-case is a core premise of data-oriented design. He also shows the promotion of deep domain knowledge in his example by storing all animation data in different arrays based on the type of data that is animated. This approach leads to good cache utilization, as a result of data being stored and processed contiguously and sequentially. Similarly, he argues for using existence based predication to avoid branches, which can be done by storing data separately based on its state.

Further on, he argues for providing an object-oriented Application Programming Interface (API) for managing the animations, with one of the reasons being that it is very easy to use. However, rather than giving users of the API an object, they can be given handles used to identify a specific animation. This approach simplifies the code and lifetime of the animations, and this simplification is a "staple" of data-oriented design.

He argues that the aim of data-oriented design is to maximize performance, but state that it also improves other aspects, such as testability and scalability in relation to multiple threads. However, he notes that while quick modifications are easier in the object-oriented case, the extra effort required for modification in DOD is worth it, as there is a smaller chance of a system turning into an overly complicated system. Furthermore, he highlights that finding the correct data separation can be difficult, that programming languages (C++ in this case) sometimes works against you, and that it can be difficult in the beginning.

> *Object-oriented programming is not a silver bullet, although it has been sold like this for a long time, and neither is data-oriented design. It's probably a combination of [the] two, and please use your best judgement because we need better software.* – (Stoyan Nikolov[45, 52:30])

Comments

As with the others Nikolov[45] state that DOD has existed for a while, but attributes the name and revitalization to the games industry. Furthermore, he clearly roots it in hardware, and in my interpretation, cache utilization specifically. However, from my understanding, he also approaches it from a bit higher level, stating that it is a "train of thought".

He also highlights inefficiencies of object-oriented programming, but do state that that way of thinking still has its place, and that DOD is not the answer to everything. Similarly, the object-oriented programming he criticizes are, from my understanding, not textbook OOP, but rather how he has experienced or seen codebases written in an object-oriented style grow.

An interesting aspect of Nikolov[45]'s presentation is the fact that he brings data-oriented design to a problem outside of the traditional game examples. His statement about deep domain knowledge, focus on common usecase, and designing for many, leads me to believe that acting on knowledge and statistics about the data also is included in his view on DOD, similar to Fabian[11]. Furthermore, the notion that it is difficult in the beginning and needing to unlearn some things, suggests to me that DOD is not simply something that can be introduced quickly, but that requires a maturation process and practice.

Christer Ericson

Ericson has presented the topic of memory optimization through industry conference presentations[46] and a chapter in his book Real-Time Collision Detection[47, chap. 13]. While Ericson does not use the term data-oriented design in these works[23], his views are discussed based on the contents of his work; memory optimization, his discussion on DOD with Acton[48], and on the fact that his presentation is linked in Bartolini's list of data-oriented design resources. My interpretation and impression is therefore that when he is discussing memory optimization he is at least partially discussing data-oriented design.

Ericson[46] motivates the need for memory optimization by highlighting how memory speed have not kept up with the increase in CPU speed. While cache is supposed to be a mitigation factor against this, Ericson claims that it is "*... under exploited in typical code*"[46, 03:27]. He further argues for memory optimization by noting that SIMD instructions consume data much faster than non parallel instructions and therefore requires data even faster. Proebsting's law[49] regarding compiler optimization improvements and that Moore's law[50] is not directly applicable to game consoles (due to fixed hardware), are further justifications for focusing on memory optimizations.

Ericson's[46] presentation mainly focuses on data cache optimization as opposed to instruction cache. He covers techniques such as prefetching and preloading to get better data cache utilization. He also suggests reordering member variables in a structure based on their use and their size due to alignment padding, or splitting a structure into two parts (hot/cold splitting), one for the frequently accessed data (hot) and one for the infrequently accessed data (cold). These can either be accessed through pointers to cold data, or the hot and cold data can be stored separately in different arrays, and accessed using an index.

Ericson argues for thinking carefully of how your data is used and choose your layout based on that, with the classical example being AoS vs SoA. "*... fields that are likely to be accessed together should be stored together*"[46, 24:14]

Ericson also brings up the issue of aliasing in C++ code, which he attributes to the use of pointers, global variables, and class members. The issue with aliasing is that it forces the compiler to be pessimistic when generating code, adding in potentially unnecessary load and store expressions

[23]These works are from before Llopis wrote his article[18], meaning that DOD had not yet received its name (assuming Llopis coined the term data-oriented design)

and remove possibilities for reordering. These pessimizations can also affect the data cache by storing back unnecessary data to the cache.

Class members are virtually as bad as having global variables as far as optimization goes.
– (Christer Ericson[46, 49:19])

He suggests mitigating aliasing by techniques such as copying function parameters into local variables and use the local variables in calculations before storing them in the output parameter, or utilizing the restrict keyword. With this you explicitly take advantage of the fact that you as a programmer know what the input and output parameters of a function are, which is not something a compiler necessarily knows. Other tips for avoiding aliasing include preferring passing small variables by value, using local variables, and declaring variables close to where they are used. The issue of aliasing is compounded by higher levels of abstraction. When data and operations are hidden through functionality such as classes, it hinders the compiler's ability to understand what data is touched, often forcing it to re-read potentially untouched fields. While some of these aliasing issues can in theory be solved by a smart compiler, my understanding of Ericson, is that such is often not the case in practice.

While his talk mainly focuses on data cache optimization, he does mention tips for code cache optimizations. These include reordering functions, moving away from encapsulation and object oriented programming, and writing monolithic functions. He notes that this traditionally goes against "good software" engineering practice, and that code cache optimizations are a moving target and should therefore be done later in a project. Additionally, he suggests keeping functions simple and have multiple versions of functions with different features rather than putting all in one function.

Avoid putting lots of features within a single piece of code. Most of the time you can split that up into two or more versions of the same function call, because you know at the caller side which features are interesting – (Christer Ericson[46, 14:23])

In an exchange with Acton[24] regarding whether or not DOD is a modelling approach Ericson[48] calls DOD a methodology or mindset:

As Mike [Acton] has eloquently pointed out elsewhere, computation is a transformation of data from one form into another.

DOD is a methodology (or just a way of thinking) where we focus on streamlining that transformation by focusing on the input and output data, and making changes to the formats to make the transformation "as light" as possible. (Here there are two definitions of "light." Mike [Acton] would probably say that "light" means efficient in terms of compute cycles. I would probably say "light" means in terms of code complexity. They're obviously related/connected. The truth might be in between.)

I say this is the opposite of a modelling approach, because modelling implies that you are abstracting

[24]See page 26

> *or not dealing with the actual data, but in DOD we do the opposite, we focus on the actual data, to such a degree that we redefine its actual layout to serve the transformation.*
>
> *DOD is, in essence, anti-abstraction (and therefore not-modelling).*
>
> *In practice, we find a balance between the anti-abstraction of pure DOD and code architecture component needs.* – (Christer Ericson[48])

Comments

Ericson has a large focus on memory and cache optimization, which makes sense given that his work reviewed here focuses specifically on memory optimization. I would however argue that his notion of thinking about data layout based on use, highlighting of the memory inefficiencies of OOP, and encouragement for understanding hardware and code generation, in my view, suggest that he is indeed discussing at least partially data-oriented design.

In my interpretation, the comment regarding multiple versions of a functions and his view of DOD as a methodology, suggests the same high-level line of thinking of utilizing what you know about your data, as that suggested by Fabian[25] and Acton[26]. In my view, another argument for this more high-level line of thinking is how he suggests taking advantage of the knowledge spatial structure of a kdtree to reduce memory usage.

From my understanding, when discussing the hot/cold splitting and accessing said elements through indices, Ericson is suggesting something akin to storing variables on a per property basis, which has also been brought up by other authors.

Tony Albrecht

Albrecht discusses data-oriented design through blog posts and industry conference presentations.

He claims that "*excessive encapsulation*"[3, p. 20] hurts performance on modern hardware, and that "*Data flow should be fundamental to your design (Data Oriented Design)*"[3, p. 20]. Both the initial presentation[3] and its revisit[51] focuses on optimizations of a scene tree, from the viewpoint of memory access. In this example[3] he investigates issues such as the cost of memory access and branch misprediction and how they affect performance.

He highlights how "optimization patterns" such as dirty flags are actually pessimizations in cases where a branch misprediction costs more than redoing a calculation, and how custom allocators ensuring contiguous memory can greatly improve performance. He further goes on to show how you can infer information from a problem, in this case that it is known that the scene tree is five levels deep, and that, from my understanding, the number of children each root has doubles per level. With this information he suggests storing the nodes in different arrays based on which level in

[25] See page 29
[26] See page 26

the hierarchy they are located in and process them in two passes instead of the original alternating single pass for better cache utilization. Furthermore, he recommends making "*processing global rather than local*"[3, p. 73], arguing against having update member functions (virtual functions in particular) in objects, also stating that it is easier to understand the code when it is all in one place.

In the end he argues for decoupling data and code from objects if possible, and that both data and code should be kept homogenous, for example by sorting by exceptions and variations rather than testing for them, and aiming for simplicity. He notes that in video games, you as a developer have control over the input data, and can therefore preformat it to reap benefits. Furthermore, he suggests having knowledge of how compilers and hardware acts, and that in general you should design for specifics, not generics.

> *... a compiler has to deal with the general case and as such will never produce optimal code for your special cases. You as the programmer know far more about your system and its data dependencies than a compiler ever could and this allows you to either provide hints to the compiler or, and optimise your code and data to allow the compiler to process it quickly. An "automatic parallelizing compiler" may give you an initial boost in performance but it will never provide you with the performance that a smart coder can extract manually.* – (Tony Albrecht[52])

Comments

From my understanding, this early presentation[3] was quite influential, but I unfortunately only have access to the presentation slides. My interpretation however is that while Albrecht's presentation mainly focuses on aspects such as branch prediction and cache misses, he still operates with the mindset of understanding and taking advantage of the data that exists within the problem.

Balázs Török

Török discusses data-oriented design in a segment of CppCast[27], a podcast aimed at C++ developers.

Török[53] notes that in his experience there is a lot of confusion surrounding data-oriented design. He describes data-oriented design as "*a different approach to problems*"[53, 27:13], and states that the core principle of data-oriented design is that "*programs are basically just transforming from data to data*"[53, 27:30]. With this principle, the sole purpose of a program is to transform data, meaning that a programmer needs to understand the data to create a good application. Similarly, a core aspect of data-oriented design is reasoning about code in terms of performance. By being familiar with the data, how it is transformed throughout the program, and knowledge of the hardware, you can reason about the expected performance characteristics of the program. Furthermore, knowing how your data is used, and what hardware you are running on, should guide the layout of the data.

[27] http://cppcast.com/

Török attributes the popularization of DOD to Acton[19] and notes that this way of thinking originates from developing on game consoles that required more low-level thinking. Having to work on those consoles led developers into a different mindset; focusing on the low level, rather than the abstractions found in object-oriented programming. He notes that this abstract thinking, while it has its benefits, can limit performance. Furthermore, it is problematic to apply the data-oriented design way of thinking everywhere, as there are cases where it is not necessarily the best solution. He also notes that DOD is probably not the best way to start for someone new to the games programming field. He brings up that the way data-oriented design is presented might be problematic, as it often focuses on aspects like not having virtual functions, base classes, etc. such a presentation can confuse and paralyze beginners.

> *The problem is, it [knowing your data] makes total sense, and everyone agrees with it, but you cannot start making your toy game engine or whatever, because the requirement is to know your data, and you do not have any.* – (Balázs Török[53, 44:14])

He argues that object-oriented programming is popular for a reason, being that it is often intuitive to think in terms of things, rather than abstract data transformations. Additionally, a big part of the requirement for data-oriented design is that you know your data, and knowing that data in the beginning can be challenging. He notes that practitioners of DOD already have an idea how their data will look based on experience.

The cases where DOD can be beneficial are cases where you have large amounts of data which you are doing smaller transformations on, as the cache will be utilized better. He exemplifies a data-oriented way of thinking through a game world containing boxes with some attributes like positions, colors, names. He argues that attributes should be stored together in collections on a per attribute basis. For example, the positions should be stored together in an array, so that when you need to update the positions of the boxes, you can iterate through only the positions, and not touch the other data belonging to a box. However, he also brings up that if you frequently need all the data of an entity when you are processing them, then you should not choose such an approach, but rather store the attributes together.

Comments

Török presents data-oriented design as very powerful, and in my interpretation, a simple concept that is difficult to master. The notion that DOD can be problematic for beginners also makes sense. They do not yet have enough experience to really know how their data looks. It can also be easy for a beginner to simply accept what industry practitioners preach, without necessarily understanding why those statements are made, or if they are applicable in all cases. This I think can be a contributing factor to the confusion that Török presents, at least that is how I am familiar with such confusion.

While he does not really go directly into the origin of DOD, it is interesting that it became a concept based on necessity when dealing with the hardware at the time. This further argues that DOD is something that has spawned from industry engineering practice, rather than academia.

David Davidović

Davidović explains DOD through a blog post, focusing on game engine design.

Davidović[54] highlights the difficulties of designing for parallelism, and bad memory access patterns of object-oriented programming as motivation for data-oriented design. He defines defines the core of data-oriented design as "*construct your code around the data structures, and describe what you want to achieve in terms of manipulations of these structures.*"[54]. Davidović states that while it is only lately that data-oriented design has been given a name, the ideas are older, and that parallels could be found in the words of Linus Torvalds[55].

> *I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships.* – (Linus Torvalds[55])

In my understanding, Davidović argues for dropping the object-oriented philosophy inside the performance critical aspects of the code that are isolated, in that you as a programmer know exactly what data they are working on. These systems can of be broken down into different steps and be viewed as processors of input, resulting in some output. Furthermore, he argues for better utilization of caching (both in terms of instructions and data) by having predictable data access and processing. A way to achieve this would be to separate data based on its state, avoiding unnecessary "if" checks that negatively affects the branch predictor.

> *Processors love locality of reference and utilization of cache. So, in data-oriented design, we tend to, wherever possible, organize everything in big, homogenous arrays, and, also wherever possible, run good, cache-coherent brute-force algorithms in place of a potentially fancier one (which has a better Big O cost, but fails to embrace the architecture limitations of hardware it works on).*
> – (David Davidović[54])

He highlights easier parallelization and vectorization, as well as easier unit testing as benefits of DOD, and state that DOD can be used together with object-oriented programming.

Comments

Davidović, from my understanding, presents a much more operational view of data-oriented design, focusing more on the implementations of data-oriented design "patterns", but does not really delve into the more high-level and philosophical views such as those of Mike Acton[28]. This is also reflected in his focus on data structures in his definition. However, the line of thinking that data is shuffled from one data transformation to the next is clearly there. The notion of brute forcing problems is not one I have seen discussed a lot in the other industry literature. The link to Linus Torvalds is interesting, but it is hard to say if he is talking about the same concept.

[28] See page 26

4.5 Literature Analysis

Based on the literature I have constructed a table of the topics discussed within the reviewed academic and industry literature. Aggregated versions of this table will be presented throughout this section, however, the full tables can be seen in the appendix (Appendix N, Appendix O). The table was created in an iterative systematic manner. As I reviewed the papers I would note down all new topics within the table as I met them, and mark the specific topic as directly mentioned (D). After the first read through I iterated on the literature, studying it multiple times, to minimize the chance of missing topics. I also merged together instances where it was clear that the different authors were discussing the same topic. Additionally, I added the classification of implied references (I), which were used in the case where something was not directly mentioned in text, but in my interpretation, implied based on the writing within its context. Lastly, the different topics were divided into a set of meta categories: origin, motivation, benefits & drawbacks, patterns, hardware related, code view, data knowledge, and philosophy. These are all discussed in later sections.

Creating the aggregated tables was done by counting how many times each topic came up across the academic and industry literature. Both direct mentions and implied references were counted as a mention. As with the literature discussed above, the aggregate count is done on a per author basis. For example, if Fontana et al. mentions a concept in each of their three papers, that is still just counted as one instance. This decision was made because I am more interested in the holistic line of thinking shown by the different authors, rather than looking at papers that just analyze one particular aspect. It should also be clear, at least from the industry literature, by now that there might be a more high-level philosophy behind data-oriented design. I would argue that this philosophical view is best highlighted when the authors are discussed holistically, rather than just their analysis of one particular aspect. All the reviewed academic and industry literature was included, with the exception of Sharp[22]. Sharp was left out because he was originally discarded in the literature filtering process, and I would argue that he is discussing a separate definition of the term data-oriented design.

While the summary of Fabian's[11] book only covered the first chapter, his book is considered in an extended sense for the purposes of DOD related topics (patterns, motivations, benefits & drawbacks). This decision was made because several chapters in the book are literally titled the name of a topic/pattern, or because they on a light skim obviously tackle that topic/pattern. However, no new topics were introduced this way to avoid introducing too many topics.

4.5.1 Weaknesses and Limitations

While I would argue that the process described above is the correct one for analyzing the content in the desired manner, it does carry with it some weaknesses.

Inflexible Direct Mentions

The direct mentions category is too rigid the way I treated it. If a concept is mentioned in text, even briefly I effectively had to include it as a direct mention. The problem occurs when a piece of literature mentions an aspect briefly, but does not delve further into that aspect. For example, multiple sources mentioned that in data-oriented design there is a focus on data rather than code. Many of those sources did not go further into that notion, or gave enough context to really understand what they meant with it. However, they still had to be marked as having directly mentioned a concept. Additionally, the direct mentions category was too inflexible, as authors who were obviously implying a particular aspect would not be included. This exclusion was the main reason behind introducing the implied reference category.

Interpretation & Lack of Context

Both the direct mentions and in particular the implied reference labels are built on my interpretation of the text. Interpreting text opens up the possibilities of erroneous labeling, both in terms of false positives and false negatives. This was particularly problematic where little space were given to describing the concept of data-oriented design, which was most of the time in the academic literature. The reason there is much more comments and discussion within the industry literature section than in the academic section is simply because the industry spent more time on describing the concept of data-oriented design. In a lot of the academic literature, the concept of data-oriented design was only given a single paragraph description.

A related issue is the lack of context in the academic papers. It might be that some of the authors discussed topics elsewhere in their works that they consider part of data-oriented design, however, there was not enough context for me to understand that they were tying those topics to DOD in particular. In the industry presentations there were usually clearer cues to what was related to data-oriented design and what was not.

4.5.2 Relative Mentions

A graphical representation of the aggregated mentions of each topic is presented in Figure 2 and Figure 3. This is to give an overview of the differences between the academic and industry literature and the topics they cover, before delving into a more detailed per topic discussion.

Description of Data-Oriented Design

Figure 2 shows the relative aggregate mentions of each topic within the categories describing data-oriented design. It should be clear from this figure that the industry has a much wider understanding of DOD than simply being about cache optimization, however, that is effectively all of what the academic literature focus on.

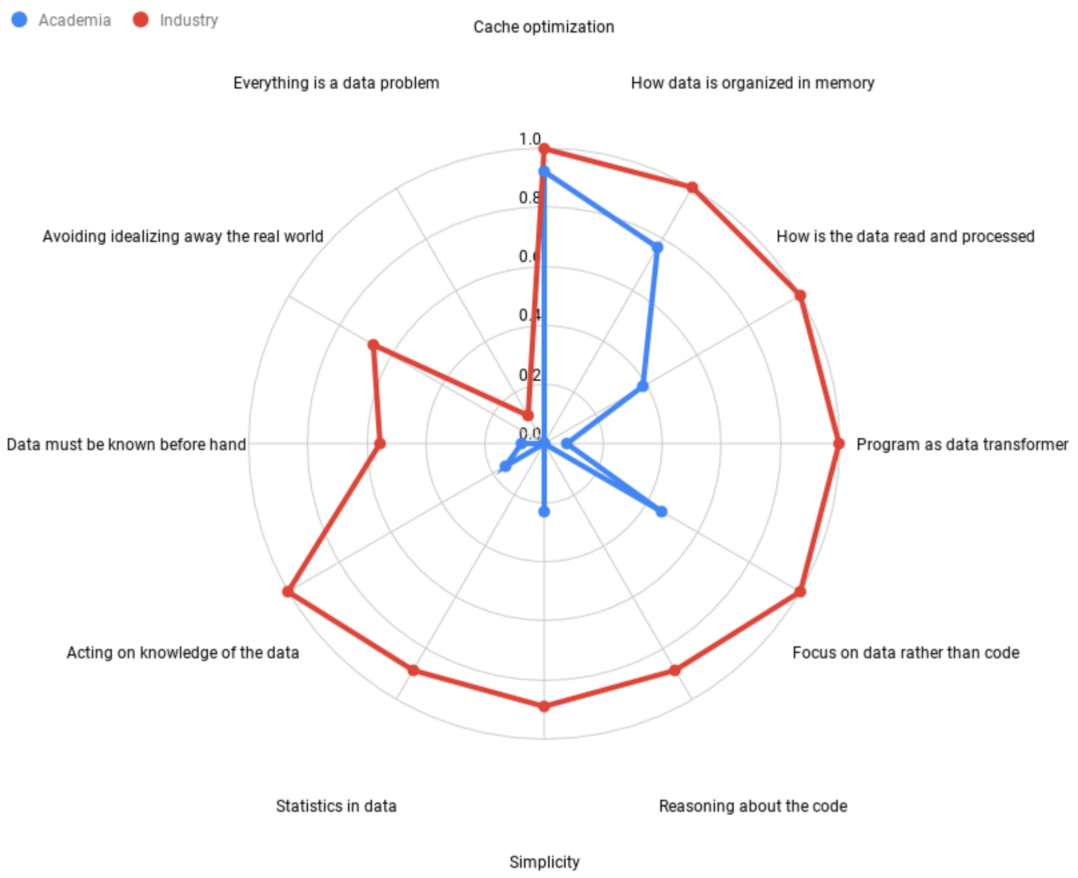


Figure 2: Percentage of Direct and Implied Mentions of various topics when describing DOD

Patterns, Motivations, Benefits & Drawbacks

Figure 3 shows the relative aggregate mentions of each topic within the categories patterns, motivations, benefits & drawbacks. The topic of origin is excluded because it does not make sense in such a visual representation. Here there are also discrepancies between the academic and industry understanding.

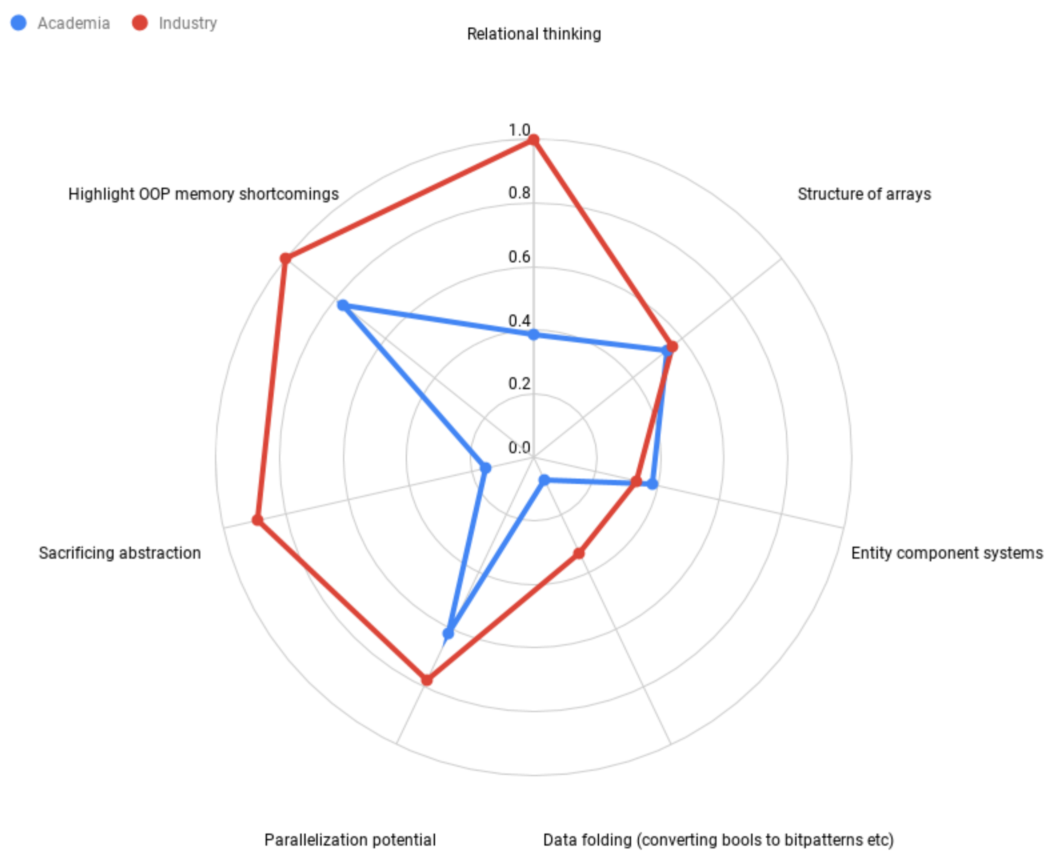


Figure 3: Percentage of Direct and Implied Mentions of various DOD related topics

4.5.3 Origin

Cited DOD Origin	Academia	Industry
Llopis	2/13	1/9
Sharp	1/13	0/9

Table 3: Aggregated count of who the different authors attribute DOD to

Table 3 shows that out of the thirteen academic authors, only three went into the origin of DOD, with two attributing the term to Llopis, and one attributing it to Sharp. Only one of the industry authors delved directly into the origin of DOD, attributing the name to Llopis.

Neither the reviewed academic nor industry literature digs deep into where data-oriented design originates from. In the academic literature Sharp[22] and Llopis[18] are the two attributed for coming up with the term. Similarly, Llopis[18] is attributed in the industry, making it reasonable to assume that he termed the concept.

As stated by Fabian data-oriented design has been around for a while in different forms. From my understanding these forms could for example be: the cache optimization aspect, SoA vs AoS, and reordering based on use, which can be traced back to at least as early as 2003 with Ericson[46]. Similarly, thinking of programs as data transformations, data being more important than code, statistics about the data, and solving the problem one has now, can be seen at least as early as 2006 with Acton[37]. Roathe[56] states to have been doing this since the early 1980's in relation to Llopis' article, however, I have not been able to find anything else to properly validate that, other than everyone saying that the ideas are not new.

Török[53] notes that the data-oriented way of thinking originates from console game development. This makes sense, as the area has characteristics which promotes this way of thinking. Console game developers know what hardware their games will run on, and how that hardware functions. Assets are often created specifically for a game, meaning that developers can adapt the detail level to fit within the constraints of the target platform. Games have clear performance requirements. At a high level a game needs to run at interactive frame rates to function at all and drops in frame rate negatively impacts the players experience. Big budget games are also expected to continue to improve in that there is no upper roof of quality, there is always room and desire for more gameplay systems or higher fidelity graphics. Game developers also have a great deal of predictability within their systems. User input might be the only unknown factor in a game at run time, and the user input is still restricted and clearly defined. Releasing a big budget title also entails spending a lot of money on marketing, which is often focused around when the game is released. These release dates are known in advance, and moving those release dates are often less than ideal. Given this description the console game industry is a natural place for data-oriented design to grow. There are hard constraints in terms of real-time requirements and release deadlines, yet there are so many known factors, so much data, that can be taken advantage of to meet those constraints.

I find it interesting that the industry highlights how the concepts are not new, yet in the academic literature it is mainly presented as a new concept. Like Fontana et al.[21] calling it a modern software technique.

Attribution

Llopis is only directly mentioned in one instance of the academic literature, with that instance being a bachelor thesis. The academic literature in general does not do a good job of attributing their sources for the DOD concept. This is particularly surprising in relation to the published articles, as these are supposed to be written by professional academics, and properly attributing your sources is an important aspect in that regard. Fontana et al. does not cite any industry literature as I can see outside of Nystrom[57] for their ECS, and while they attribute DOD to Sharp[22] I have not really been able to find the DOD they describe in his work. Similarly, I have not been able to find attributions to Sharp in the industry literature.

Lietchy does not cite any industry literature at all, while still giving an overview of what data-oriented design is, and in Lietchy's case this is much more problematic. The astute reader might have noticed similarities between Lietchy's and Llopis' descriptions of DOD, this is because Lietchy appears to have plagiarised Llopis' work, and in my view this was done intentionally, as small modifications to the text has been done to fit it into Lietchy's content. The comparison between these two descriptions can be seen in Appendix A.

4.5.4 Motivation, Benefits & Drawbacks

Topic	Academia	Industry
Highlight OOP memory shortcomings	10/13	9/9
Parallelization potential	8/13	7/9
Sacrificing abstraction	2/13	8/9

Table 4: Aggregated count of topic mentions of motivation, benefits and drawbacks related topics

Both the reviewed industry and academic literature spends time motivating data-oriented design by highlighting the inefficiencies of object-oriented programming, particularly the cache utilization aspect. Easier multi-threading potential is also brought up as a motivation for moving to a data-oriented design approach. Similarly most of them state that data-oriented design is different from object-oriented programming.

Opposition Against OOP

Romeo[33] draws attention to common complaints that DOD is less convenient and safe, and that encapsulation and abstraction is sacrificed to gain performance and flexibility. From my findings, these beliefs are not directly reflected in the rest of the reviewed academic literature. However, a lot of the literature both in the industry and in academia dedicate time pitting object-oriented programming against data-oriented design. The level of aggressiveness and which aspects they tackle varies. From my view, the academic literature mainly echo the statements of the industry in this aspect. A reasonable question to ask is why there seems to be so much opposition against object-oriented programming in DOD. In my understanding and opinion, there are several reasons for this opposition.

One reason is the values of encapsulation and abstraction that object-oriented design promotes. The point that OOP often leads to inefficient cache utilization by storing data that might not be processed together in the same object, is brought up by most of the literature. Related to this, in my understanding, is that excessive abstractions can make it harder to see at a glance how source code is mapped to machine level instructions. Inefficiencies such as these are obviously not ideal in an industry where performance is an important factor. Debugability in terms of seeing how the source code is executed by the machine is also perceived by some as more difficult. For example due to the mental burden of context switching when stepping into object member functions and being presented with the internal context of an object. Another reason can be what Fabian eludes to, that excessive abstractions hinders the ability to see the data in its primitive form, i.e. lists, trees, etc. even numbers, which could help a programmer make more informed decisions in how to store and process the data in question.

It should be noted that the aspects of understanding how code is mapped to machine level instructions and debugability mainly show up in the industry literature. The academic literature almost exclusively focuses on the caching aspect in that regard.

Fabian also brings up an interesting aspect in related to the "opposition against OOP", being that, in my understanding, object-oriented programming was the predominant way of developing software around the time data-oriented design entered the stage. Fabian notes that DOD was at one point fighting to be heard. From my understanding, this could potentially be due to aspects such as appreciation for hardware, separating objects into multiple containers, and so on, often went against what was seen as conventional best practices.

Functional Programming

Both the industry and academic literature bring up the notion of viewing programs as data transformations, working on inputs and outputs. As previously stated, Llopis notes that a functional language might be ideal. Similarly, Acton and Fabian comment on there being overlaps between functional programming and data-oriented design, but that they are not the same[58][29]. I have not been able to find a direct link between DOD and functional programming as a paradigm within the reviewed academic literature.[30]

My understanding, is that the overlaps with functional programming is the desire for side effect free functions, and viewing programs as mere data transformations with clear inputs and outputs. Aspects such as higher-order functions, lazy evaluation, referential transparency, or pattern matching[60] does not seem to be heavily promoted within DOD.

Fabian presents a short discussion on lazy evaluation in relation to optimization[11]. My understanding is that as a programmer you should find ways to avoid expensive computation early, such as existential processing rather than a last minute check of a dirty flag (Fabian draws a link here to Albrecht's example[31]). From my understanding, this use of the term lazy evaluation does not match the lazy evaluation that Hudak discusses[60], where lazy evaluation is handled at the language level, rather than by the programmer.

[29]Being a small exchange on a social media platform means that one should not necessarily read to much into this. However, I would argue it is still an interesting notion that warrants discussion

[30]It might be that Laine et al. discusses such a connection. However, I was unable to get a good translation of their paper. (Janne Laine, J. Y. Data-oriented design and functional programming in unity. Bachelor's thesis, Kajaanin University of Applied Sciences, 2017. Available from: `https://www.theseus.fi/bitstream/handle/10024/129313/Laine_Janne_Ylisiurua_Juho.pdf?sequence=1`)

[31]See page 37

Fabian[11] also notes that it might be helpful to understand aspects such as static single assignment form, to help the compiler make better choices when compiling your code.

In my understanding, the focus on considering the target hardware and how it works on a more fundamental level would rule out pure functional programming. Lastly Fabian[11] mentions that the language that benefits the most from data-oriented design is C++, and Acton[19] notes that ideally, they would write everything in assembly.

> *Software does not run in a magic fairy aether powered by the fevered dreams of CS PhDs. Software is real engineering, that runs on real hardware, to solve a real problem* – (Mike Acton[19, 16:11])

Any deeper discussion of the relationship between DOD and functional programming is outside the scope of this thesis.

4.5.5 Patterns

Topic	Academia	Industry
Relational thinking	5/13	9/9
Structure of arrays	7/13	5/9
Entity component systems	5/13	3/9
Data folding (converting bools to bitpatterns etc)	1/13	3/9

Table 5: Aggregated count of topic mentions of pattern related topics

As seen in Table 5, both the academia and industry discusses a set of different patterns that can be used within data-oriented design.

One of these patterns is storing properties together on a per property basis. For the purpose of this work, I label this as "Relational Thinking", as Fabian[11] discusses at large relational databases, and how they can aid in thinking more in a data-oriented manner. It can be difficult to interpret when the authors are discussing the relational thinking aspect and when they are talking about SoA, as the description of the two can overlap. I view relational thinking as more high-level than SoA structures (discussed further on in this section), as per my understanding, SoA is executed on a local, per struct basis. Furthermore SoA, at least from my understanding so far, is almost exclusively concerned with the layout of the data and their types. Relational thinking on the other hand, lifts the focus from individual structs to a more organizational view. Concerning itself also with separation based on state, conditional execution, and relationships and communication between systems.

Fontana et al.[32] and Faryabi[33] seem to view relational thinking as being the core of data-oriented design, with the motivation being better memory access, and flexibility. Grieshofer[34], Osborn[35],

[32] See page 16
[33] See page 20
[34] See page 21
[35] See page 17

and Bond[36] also seems to bring up the relational aspect, however, it is difficult to ascertain if they view it as being the core of DOD. From my understanding, none of the other academic literature reviewed in this work really goes into this aspect. Lietchy[37] touches on the per property aspect, but my understanding is more that he provides a SoA structure behind an object-oriented interface, similar to the first optimization done by Albrecht[3]. While the academic literature often mentions storing properties together on a per property basis, the wider implication of relational thinking, such as storing properties based on state to avoid branching through existence-based prediction, is usually not present within the academic literature.

In the industry literature, the aspect of relational thinking is often implied. Storing properties based on their type is mentioned by all the larger works, with most of them also mentioning sorting based on common operations or conditional execution. However, these groupings does not seem to be the core of DOD in the industry's eyes, but rather an effect of thinking of the data. For instance, Llopis[38] notes that it is just "often" that contiguous homogeneous data for sequential processing is a preferred format. Acton suggests storing state types separately as a rule of thumb[19]. Lastly, Fabian[11] notes that working in a database style is not a requirement for data-oriented design, but rather it is something that can help getting data into a format that is easier to reason about.

The structure of arrays pattern receives wide attention within the academic literature, particularly among the master and bachelor theses. However, it does not seem to be considered as important within the industry literature. Mainly being mentioned as a possible optimization by Fabian[11] and Nikolov[45]. Furthermore, there seems to be more thought behind it in the industry literature. The academic papers often highlight how the structure is cache friendly. However, as noted by Török[53], this is only efficient if you actually access the data separately. If you need access to all the data on a per entity basis, and they are stored in separate locations, you should incur more cache misses than if they were stored together. Whether you choose a structure of arrays or arrays of structures approach depends on how you will process your data, and the dependencies between them.

Lastly, entity component systems are discussed as a pattern that fits well within data-oriented design, both by the industry and academia. Fontana et al. seems to tie ECSs and DOD very close to each other, however, I have not found anyone else that ties the two concepts together as strongly.

4.5.6 Hardware Related

Table 6 shows that both the academic and industry literature dedicate attention to the discussion of cache optimization. It should be noted that there is a super-set subset relationship between the different categories in this table, meaning that all the sources that brought up "how is the data read and processed" also brought up "how data is organized in memory". While both "how data is

[36]See page 19
[37]See page 15
[38]See page 23

Topic	Academia	Industry
Cache optimization	12/13	9/9
How data is organized in memory	10/13	9/9
How is the data read and processed	5/13	9/9

Table 6: Aggregated count of topic mentions of hardware related topics

organized in memory" and "how is the data read and processed" falls under the category "cache optimization" I chose to keep them separate. From my understanding of the reviewed literature, there is a subtle difference. "How is the data read and processed" also implies knowledge of how the data is used in the program and how that affects caching. This line of thinking was not as clear in the papers focusing on the other two categories.

While both academia and the industry covers the aspect of cache misses, how much focus it gets varies. The academic literature has a large focus specifically on avoiding cache misses, with some directly stating that it is the main goal of DOD. My interpretation is almost that if the hardware suddenly changed, limiting the impact of cache optimization, data-oriented design would still be about a focus on writing cache friendly code. However, the impression I get from the industry is quite opposite. Fabian states that "*A common misconception about data-oriented design is that it's all about cache misses*"[11, p. 10], and Acton promotes the importance of understanding your hardware. My interpretation is that here, the cache discussion is an effect of having a better understanding of the hardware, rather than being the goal itself.

Different hardware, different solutions – (Mike Acton[19, 17:57])

4.5.7 Code View

Topic	Academia	Industry
Program as data transformer	1/13	8/9
Focus on data rather than code	6/13	9/9
Reasoning about the code	0/13	8/9
Simplicity	3/13	8/9

Table 7: Aggregated count of topic mentions of code related topics

Table 7 shows more disparity between the reviewed academic literature and the industry literature. It should be mentioned that the aspect of data flow or pipelines, as used by Albrecht[3] and Nikolov[45] are included under the view of program as data transformer. This is to avoid introducing too many terms, and from my understanding, these terms are overlapping in their use.

The view of programs existing solely for transforming data is more prominent within the industry literature. Closely related to this view is the notion that data is more important than code. My understanding to these aspects is that the code itself has no inherent value. It exists solely to transform data from one form to the next. With that, the data and how it is transformed should be viewed as more important than the code. Even though these aspect are mentioned directly in some academic literature, they are more clearly defined and has a much stronger presence within the industry literature. I get the feeling that code which does not add progress towards the final goal of solving a problem is considered bad practice within the industry. However, I do not get this feeling from the academic literature.

A concern that is frequently brought up in the industry that is effectively missing from the academic literature is being able to reason about the code. Reasoning about the code means understanding not just at a high level what the code is doing, but rather how the source code maps to instructions that will be executed and what assumptions where made about the data and the hardware a transformation was written for.

Simplicity is also brought up much more in the industry literature. Simplicity is obviously a wide term, however, my understanding of simplicity in this case is being clear about what you are doing at a code level, and also not doing more than what is required to solve your problem. You should choose a straight forward approach based on the information and data currently at your disposal. This might sometimes lead you to brute force solutions as those suggested by Davidović[39]. My understanding of simplicity also involves avoiding unnecessary abstractions and generic design, as these will lead unnecessary complexity by introducing concepts that might not be relevant for the problem you are currently solving.

4.5.8 Data Knowledge

Topic	Academia	Industry
Statistics in data	0/13	8/9
Acting on knowledge of the data	2/13	9/9
Data must be known before hand	1/13	5/9

Table 8: Aggregated count of topic mentions of domain knowledge related topics

As seen in Table 8 the importance of knowledge of the data data receives wide attention in the industry literature, but is largely forgone by the academic literature.

Karlsson[40] notes that "*The DOD paradigm implies that the data need to be known beforehand*"[27, p. 37], similarly, Grieshofer[41] also implies that you should act based on the knowledge you have about the data your are processing. For instance, that if some data is only rarely needed, it can be stored separate from the rest of the data. Outside of these two, the academic literature does not dedicate much attention to statistics and patterns within the data, acting on knowledge of the data (including domain knowledge), or the notion that data must be known beforehand.

[39] See page 40
[40] See page 18
[41] See page 21

The industry literature on the other hand places a greater focus on knowledge about the data. Acton[42] argues for discussions with content creators, estimates, and tools to gather information about the data to make better decisions. Fabian[43] establishes information such as statistics, shape, probability, etc, as a principle of data-oriented design. Similarly, Llopis[44] promotes the notion of an ideal data format, which implies the need for knowledge of that data. Furthermore, Nikolov[45] states that data-oriented design promotes deep domain knowledge.

From my understanding, in this view of data-oriented design, data ends up having a different meaning. Rather than just considering data from the computer science perspective (data being bytes), data is considered in a wider sense, to also include information and knowledge about the problem you are trying to solve.

> *The fundamental truth is that data, though it can be generic by type, is not generic in how it is used. The values are different and often contain patterns we can turn to our advantage.*
>
> – (Richard Fabian[11, p. 15])

4.5.9 Philosophy

Topic	Academia	Industry
Avoiding idealizing away reality	0/13	6/9
Everything is a data problem	0/13	1/9

Table 9: Aggregated count of topic mentions of domain philosophy related topics

The reviewed industry literature seems to be much more pragmatic and grounded in the reality of their domains. That reality being that they work in areas where performance is important, that they can reason about the commonalities in the hardware they need to support, and that they have a specific set of problems. This notion of not idealizing away the reality of a situation, taking resources, knowledge, and constraints (both inside and outside of the code) into considerations to find an optimal solution, is not really present within the academic literature. Rather, this seems to be a more holistic view held by some of the authors within the reviewed industry literature. From my understanding, Acton in particular seems to elevate this holistic view to almost a general way of solving problems. I base this on how he states that everything is a data problem, not just performance, and that he describes it as "*Practical philosophy not methodology.*"[42, p. 5].

4.6 Frequency of Mentions

So far we have seen the breadth of topics covered by the academia and the industry, as well as the differences between the two. An aspect that has not been covered is how many topics are mentioned

[42] See page 26
[43] See page 29
[44] See page 23
[45] See page 33

by the different authors. A graph representing the number of topics mentioned by the different authors can be seen in Figure 4. On average (arithmetic mean) the industry authors mention a total of 15.00 topics each (standard deviation=2.24), while academic authors mention 6.23 topics each (standard deviation=3.06). This should indicate that the industry has a more holistic view of DOD, and that academics might have misunderstood the concept. However, as can be seen from Figure 4 and the academic standard deviation score, there is much larger variety in how many topics are mentioned. Karlsson is the author who mentions most topics among the academics.

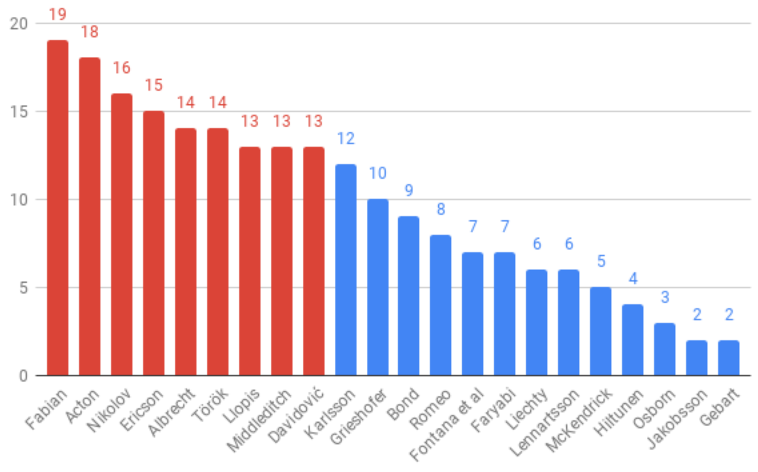


Figure 4: Total number of topics mentioned by each author, sorted highest to lowest.
Red = Industry authors, Blue = Academic authors.

I previously discussed some of the weaknesses of the direct and implied mentions model, touching briefly on the issue of interpretation. Mainly being the issue of bias and erroneously labeling a concept. While this is obviously a weakness with the approach, I would argue that it has worse consequences in relation to implied mentions. Direct mentions are more objective, based on that they either require that the topic is mentioned word for word, or with clear synonyms. However, implied mentions leans stronger on my interpretation and opens more up for bias. This is also why for transparency reasons I am supplying the full tables in the appendix (Appendix N, Appendix O).

Figure 5 shows how many direct mentions and implied mentions I recorded for each author. It is clear that I recorded many more implied mentions within the industry. However, even if we remove implied mentions the pattern that the industry has a more holistic view is still clear. On average (arithmetic mean) the industry authors directly mentions 11.78 topics each (standard deviation=3.63), while the academic authors directly mention 5.15 topics each (standard deviation=2.79).

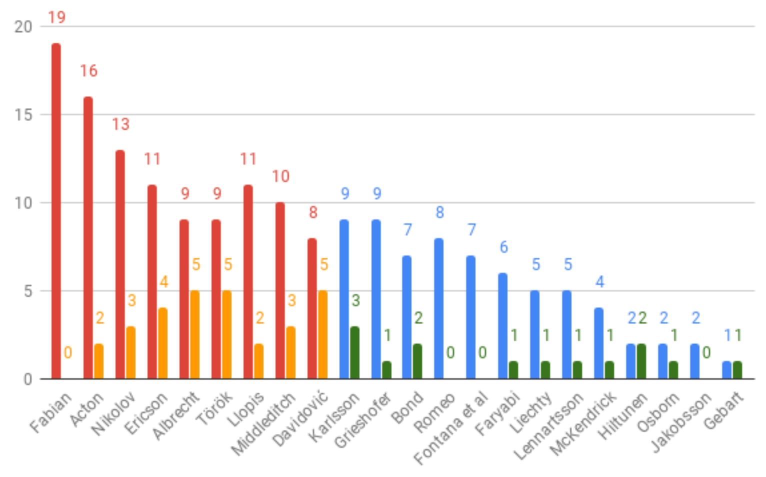


Figure 5: Implied and Direct mentions of topics by each author, sorted highest to lowest based on total mentions.
Red = Industry direct mentions, Orange = Industry implied mentions
Blue = Academia direct mentions, Green = Academia implied mentions

Due to the weaknesses and interpretive nature of the mentions I am intentionally not conducting further statistical analysis on this data. While I believe it is good enough to serve the purpose of proving the point that there is a large difference between the academic and industry understanding of DOD, it is not robust enough to say anything more sophisticated than that.

Why more implied mentions in industry?

I believe the reason that the industry has received so many more implied mentions is partly due to how much time is dedicated to discussing DOD, which medium they operate in, who their target audience is, and what language they use. Most of the academic literature have paragraph short descriptions, and also use a more formal language, usually describing *what* without discussing *why*. This is possibly due to academics being more driven by a desire for contribution rather than explanation. The short length and type of language does not lend itself well to analyze in search of an underlying way of thinking that can be used to understand if the author is implying something. This is best explained using an example.

Gebart states the following in his work:

> *The opposite to OOP is called Data Oriented Programming (DOP), the focus in DOP is to arrange the data in a way that is more efficient for the computer hardware, grouping data primarily by which attribute it belongs to instead of grouping data by which object it belongs to. This is realized by using a single struct containing each attribute as an array of some primitive data type (int, float, double etc.), this approach is called Structure of Arrays or SoA.* – (Joakim Gebart[30, p. 32])

This is the only place where the words "data oriented" are mentioned in Gebarts work, all I can do is note down that he directly mentions SoA[46], and implies cache optimization. There might be other places in his thesis that he brings up aspects that he personally think is related to DOD but that is hard for me to interpret due to the lack of context. After all, he does not mention the words "data oriented" again and his thesis is not about DOD specifically but about GPU implementations of particle filters.

Contrast this with Török who also, at one point during his appearance on CppCast[53], brings up SoA as a way that can lead to better cache utilization, branch prediction, etc. Later one of the hosts asks if an SoA approach is problematic when you need to access multiple attributes from each "object" at the same time. Török notes that that is the case, but that you need to think about how frequent you are accessing all the attributes together at the same time. If you are most frequently accessing all the attributes together at the same time, you should not be using an SoA approach.

Török is participating on CppCast specifically to discuss data-oriented design. This means that in the section of the podcast where DOD is discussed I can assume that the approaches and topics he covers are related to DOD unless otherwise specified. From this example alone I would note down the following mentions. When he brings up SoA and cache utilization I can mark those as directly mentioned. I can also mark "acting on knowledge of the data" and "statistics in data" as directly mentioned, because that is semantically what he is saying when expressing that you need to think about the frequency of accessing all attributes together at the same time. Lastly, I mark "how is the data read and processed" in relation to hardware as implied due to how Török brought up cache utilization and branch prediction when originally discussing SoA.

I can note down these mentions and implied mentions because I know that this CppCast is discussing data-oriented design, not some other concept like particle filtering on GPUs. Additionally, Török is more "open" in his description of DOD, which allows for more interpretation. Gebart is essentially saying DOD = SoA, which does not leave much room for interpretation and looking for implied meanings.

Hollow Replication

Another issue with a lot of the academic literature is that it re-uses a lot of vague phrases used in the industry, without contextualizing them enough to understand what they actually mean. When Llopis states that "*Data-oriented design shifts the perspective of programming from objects to the data itself: The type of the data, how it is laid out in memory, and how it will be read and processed in the game.* "[18, p. 43] and later contextualizes that by discussing architecting your program around an ideal data format, strongly suggests aspects such as knowing statistics of the data and acting on it. This is in my interpretation even more strongly implied when Llopis argues for transforming data

[46]My understanding of Gebart is that SoA must make use of primitive types, which is not the understanding I am operating with in this thesis

into the ideal format offline. Lietchy[20] states essentially the same regarding that DOD is about placement in memory and how it is read and processed[47]. However, no discussion on topics such as an ideal data format are given. With the little text that is given I do not get enough context to clearly say that he implies knowing and acting on statistics of the data.

4.7 Understanding of Data

My impression of the academic literature is that data is effectively the opposite of functions, i.e. a very code-centric view. However, my impression of the industry literature is that data is considered in the widest sense of the word (information, statistics, problem knowledge, etc), and with that data-oriented design also entails making decisions and being guided by data.

For example, Fontana et al.[21] finds that in one of their benchmarks, the object-oriented solution outperforms their data-oriented one. They conclude that their data-oriented solution outperforms their object-oriented one in cases where data locality can be fully exploited. However, in the case where data locality cannot be fully exploited, the object-oriented model is faster. They explain the benchmark case where the object-oriented implementation is faster as thus:

> *Problem B: estimating the interconnection wirelength for all circuit nets. This scenario accesses different properties of different entities. Therefore, it cannot efficiently explore the data locality provided by DOD, since the properties of the pins in a single net may not be contiguous in memory, unless the property array is previously sorted.* – (Fontana et al.[21, p. 30])

If having the properties sorted would imply sequential memory access that could increase performance, then why is sorting or keeping the properties array in a sorted state not discussed? Similarly, they do not discuss how common the two different problems are. In their first problem, only one property is needed per entity, and from my understanding each entity is processed sequentially. Is that the most common type of problem that their library will help solve, or is the common case that one need access to multiple properties over multiple entities? Are there a set of properties that are more likely to be accessed together? Is it pure performance they are after, or is the flexibility of having algorithms that only access the data they truly need that is the main goal?

From my understanding, these questions were not discussed, and if that is the case then I would argue that Fontana et al. misses the point of being guided by and taking advantage of knowledge about the data. This is not just the case for Fontana et al. In general most of the academic literature seem to be thinking on an operational level, rather than a holistic one. Thinking of data-oriented design as applying different design patterns, without investigating whether or not their decisions are actually supported by their data (assuming they actually collect any data).

Another possible reason for the narrow view on DOD is the usually narrow scope and decontextualization that is promoted in the academic profession through aspects such as encouraged specialization areas and page limits in journals.

[47] See discussion on plagiarism, page 46

4.8 Summary & Conclusion

Research question 1[48] focused on the differences in conception and discussion of DOD between the industry and academia. As seen in the analysis[49] and the previous section[50] there is a divide between the conceptualization and discussion of DOD between the industry and academia. While academia and the industry cover a lot of similar topics, the academic literature largely ignores the aspects covered by the code view, data knowledge and philosophy categories. The cache utilization aspect is covered extensively, as if that is all there is to DOD. It can be understandable that the academic literature predominantly associates DOD with cache and hardware oriented programming, as these aspects are often at the center of attention in the industry literature. However, academia, in my understanding, miss the line of thinking behind the examples that the industry literature is giving.

The academic literature has a more operational view on DOD, seemingly thinking more in terms of applying patterns. On the other hand, the industry has a more pragmatic and holistic view with a focus on solving the actual problems they face given their resources and constraints. The categorisation (see Appendix N and Appendix O) provided in this chapter can serve as a starting point for a clearer characterisation of DOD in the academic and industrial literature.

In conclusion and as an answer to research question 1, this chapter has shown that there is a large gap between the industry and academic literature in relation to the conception, discussion, and description of DOD.

[48] See page 1
[49] Section 4.5
[50] Section 4.7

5 Interviews & Characterisation of DOD

Based on the previous summary of and discussion surrounding the academic and industry literature, it should be clear that there is more to data-oriented design than just cache optimization. The industry seem to have a more holistic view on data-oriented design. To further refine data-oriented design and reduce the chance of misunderstanding the topic, I decided to conduct a set of semi-structured interviews[8] with industry practitioners. The goal of the interviews was to capture the practitioners understanding of data-oriented design in order to find commonalities and generate discussions. As with the literature review, the analysis of the interviews are highly interpretative. As such I will be liberal with the use of direct quotes from the interviews in order to be more transparent. The quotes themselves are directly copied and pasted from the online documents used for the interviews, while manually keeping the original formatting (bold type, italics type, bullet points, etc.). This means that any grammatical errors in the quotes are by the participants. I will not make note of those errors; i.e. "[sic]". Doing so, in my view, both detracts attention from the participants answer, and is unfair for participants who are not as well versed in English, as they would have more of their quotes covered with notes. Additionally, the level of formality varied between the participants, and the interviews often had quite a casual nature.

Lastly, several of the quotes will use DoD as an acronym for DOD. From my understanding the canonical capitalization is DOD, however, at the time of conducting the interviews I was capitalizing it as DoD. This could have influenced the capitalization used by the participants.

5.1 Conducting the interviews

5.1.1 Participants

The participants was selected amongst the authors behind the various resources on data-oriented design organized by Bartolini[1]. However, I restricted the group to authors who was explicitly talking about data-oriented design, leaving out authors behind works that was not discussing data-oriented design by name, or by "intent". This is simply a result of Bartolini's page containing resources where it is unclear whether or not it is a resource directly on data-oriented design, or simply orthogonal concepts that are useful in a data-oriented design setting. Since I have been interested in this topic for a while, I also included industry professionals behind literature outside of Bartolini's list, or professionals with whom I have previously discussed the topic. In total I contacted 13 industry professionals, out of which 8 participated in the interview. No forms of compensation or incentives for participating were given to any participants. The participants were also asked to indicate whether

[1] https://github.com/dbartolini/data-oriented-design

or not they could be quoted in the thesis, either anonymously or by name. One participant was identified as an outlier and removed, this was due to limited recent exposure and practical application of DOD.

Raw Interviews

After conducting the interviews I discovered that I had gotten much more data and greater insights than I originally anticipated. Because of this I went back to the participants and asked for permission to publish the interviews. My personal recommendation is to go and read the interviews in their full form, as there is more wisdom there than I can properly reflect given the scope of the thesis. The full interviews of the participants who allowed me to publish their interviews can be found in the appendix (Appendix D, Appendix E, Appendix F, Appendix G, Appendix H, Appendix I). I would personally recommend reading through the interviews, as they give interesting insights, all of which I am not able to cover in this chapter alone.

5.1.2 Structure

The interview followed a semi-structured form, and was done through the use of an online document. Interviewing the professionals through the online document circumvented the problem of coordination with the participants located in different time zones. Furthermore, it also allowed them more time to think thoroughly about their answers, and gave me the opportunity to ask follow-up questions, or ask for clarifications to minimize chances of confusion.

Privacy and The Norwegian Centre For Research Data

Privacy became an issue both due to the usage of online documents, and because I was recording the participants names for the use in quotes. To avoid directly tying the participants names to their respective documents each participant was given a numerical identifier, which was stored together with their name, in a separate file. The interview documents were hosted on NTNU's sharepoint solution and participants were given auto generated links to access their respective documents. The Norwegian Centre For Research Data[2] (NSD) was notified, and approved this approach[3].

5.1.3 Questions

The set of initial questions and the follow-up questions that were asked of all participants can be seen in Appendix B. The questions covered a range of topics, from the participants experience with data-oriented design, their principles and approach, as well as benefits and downsides. All of the questions were written with the intention of elaborating on a certain topic, which can be seen in Table 10. The reason for asking multiple questions regarding the same topic was done in an attempt to "triangulate" an understanding of the various topics. However, it was not uncommon

[2] https://nsd.no/nsd/english/index.html
[3] Reference number: 444859

for participants to touch upon other topics than the one that was initially intended for a specific question. As a result I viewed the intentions of the questions as a guideline, rather than strict rules for what to include under which topic. For example, the topic of the participants understanding of DOD often appeared in what they considered misconceptions about DOD, or hurdles of learning DOD as well. The questions covers a wide range of topics both in an attempt to investigate DOD from multiple angles, and to highlight potential areas for future work.

Question	Intention
Question 1 & 18	Experience & Confidence with DOD
Question 2-6	Understanding of DOD
Question 7	Origin of DOD
Question 8-9	Where to apply DOD
Question 10-15	Benefits, Drawbacks, & Productivity
Question 16-17	Hurdles of Learning DOD & Misconceptions

Table 10: Overview of intention of interview questions

5.2 Interview Responses

5.2.1 Experience with DOD

Around half of the participants had at least known about data-oriented design for about 7-9 years. Meaning that these participants first learned of the concept after it was named. The other part of the participants all state to have worked with the concept for over 10 years, with the longest being 19 years.

An interesting point made by two of the participants was that it took a while from when they first learned about the concept to when they were able to understand and utilize it successfully. Dale Kim and Marc Costa stated the following when asked how long they had been working with a DOD mindset:

> **Dale Kim –** Trying for 7 or 8 years but only really successfully doing it for maybe 4 years.
>
> **Marc Costa –** I learned about DOD around 2011 or 2012. Developing the mindset took years, though, since resources about DOD were very scarce and, after all, it's exactly this, a mindset, and not a set of design rules.

While all participants stated that they had known about DOD for several years, most participants only ranked how confident they were in their understanding of DOD as a 5 on a 1-7 Likert scale (with 1 being not confident, and 7 being very confident). The exception being Richard Fabian, and Stoyan Nikolov who rated themselves as a 7, and 3 respectively.

5.2.2 Understanding of DOD

As previously mentioned, questions 2-6 were all meant to extract an understanding of DOD from the participants in different ways. For example, the question "What does DoD mean for you?" was meant to allow the participants to freely retell how they understand the DOD concept, while the principles of DOD would require them to nail down the priorities of the different concepts. The answers from the participants were quite varied, but I extracted some common themes that were clearly present. These are discussed below.

Hardware Knowledge

Understanding cache utilization was heavily discussed within the literature, and the topic is also present within the interviews. However, rather than just focusing on cache utilization, most participants promoted an understanding of hardware and the consequences of hardware in general. Balázs Török, Stoyan Nikolov, Dale Kim, and Marc Costa all brought up hardware knowledge as principles of DOD. The latter three elaborated a bit on their perspective on hardware, with all of them, from my understanding, clearly presenting a view of hardware as a resource problem. The point brought up by Dale Kim regarding how the details of a language is not a concern as long as you have sufficient hardware access is particularly interesting. In my interpretation, it solidifies the fundamental view that as programmers our only goal when writing code is technically to manipulate hardware into a state where our problem has been solved. With this view, knowledge of the hardware should obviously be embraced, because it is a fundamental part of the programming field.

> **Balázs Török –** To solve a data transformation problem, you can't disregard the hardware limitations
>
> **Stoyan Nikolov –** Platform -> The technical details of the hardware that will be running the program matter a lot. The designer should embrace this knowledge and make use of it, instead of hiding it and pretending she is working on a totally abstract machine. While there certainly are differences between platforms –PC, mobile, consoles etc., there is significant overlap we should not ignore, i.e. memory access is almost always quite slow.
>
> **Marc Costa –** DOD is a mindset that makes you look at programming from the bottom up: considering the hardware first and solving problems with that in mind. The hardware capabilities are the physical set of limitations we have to work within. Playing to the hardware strengths guarantees that the software produced will run better (e.g. better runtime performance, less memory used) than software ignoring or working against the hardware strengths.
>
> **Dale Kim –** [DOD means] Having as complete of a knowledge as possible of the hardware and how the data should flow to maximize performance on that hardware. Which language you use and the details of those language are almost inconsequential so long as you have adequate access to the hardware and how memory is managed.

Dale Kim, Balázs Török, Marc Costa, and Stoyan Nikolov also included hardware understanding in what they consider the principles of DOD. From my interpretation, Stefan Reinalter does include understanding memory and cache as aspects of DOD but not as strongly as the others. Tony Albrecht promotes knowledge about hardware as a good thing, but not a requirement for doing DOD.

> **Tony Albrecht** – Understanding the way your data is being used (or is to be used) **would** be part of DoD. You **do** need to understand how your data is to be used in order to apply DoD. Knowledge of the HW you are running on is necessary for efficient execution, but not for DoD practices.

Simplicity

Simplicity showed up in the literature section, and is also mentioned by the interviewees. I originally interpreted simplicity to mean not doing more than what is required to solve a problem, and being clear about what is done at a code level. Such an understanding is also how I interpret the interviewees. Tony Albrecht, Stoyan Nikolov, and Marc Costa bring up simplicity directly in their principles of DOD, and in my understanding so does Dale Kim.

> **Tony Albrecht** – One of the primary implications of DoD (for me) is performance. A primary side effect is the often complex logic required to solve problems can be dramatically simplified with DoD. If I see a complex set of classes, layers of inheritance, friend members in classes, mutables, and other OO hacks in code, I'll take a step back and look at how the dfata flows through this – what goes in, what comes out. Then, throw away all the code after the input and before the output and build something that does just what it needs to do. And only what it needs to do. Complexity often arises from solving problems that have evolved over time, or from not really understanding the basic issues at hand. Complexity in a DoD solution implies that the core problem may not be well understood – DoD applications should consist of relatively simple operations on data. Of course, those simple operations can be complex in and of themselves. Fundamentally, what I think I'm saying is that complexity is an indication of bad implementation, regardless of paradigm. All programmers should strive to produce code which is as simple as possible.

> **Marc Costa** – Simplicity: I'm a big proponent of simple solutions to problems. DOD focus on the data transformations, instead of code architecture. By shifting the focus, programmers are prevented from creating massive amounts of code in the name of some abstract goals like abstraction, clean code, generality or extensibility.

> **Stoyan Nikolov** – Simplicity -> Not sure if this is DoD or general Software architecture, but too often developers corner themselves with over-complicated designs and tons of abstraction layers that have little to no justification. DoD, by forcing us to know which data is needed where, can simplify our designs.

> **Dale Kim** – Set up data structures and data flow so that the final data is achieved as directly as possible (no unnecessary data transformations or excessive maintenance of state that's not relevant to the problem being solved).

The related aspect of minimizing state was also brought up by other participants. Stefan Reinalter in particular heavily promote pure functions, and Richard Fabian also brings up the aspect of minimizing state.

> **Stefan Reinalter** – Prefer pure functions to everything else – those functions don't touch state, don't have state, and get passed all required input and output parameters.

> **Richard Fabian** – Complexity and bugs come from unexpected state, so where possible, make state highly visible with paper trails where you can, and also reduce the amount of state you carry around when possible. Separate real state (some would call it the model) from the state you create to help solve the problems (some might call this viewmodel, or accidental state, or even transitional data, but it's all the same in essence, it's state which is only necessary because of how we are solving the problem and is not intrinsic to the model)

Data Qualities

All participants brought up the importance of having an understanding of how data was accessed, and patterns within the data, and, in my interpretation, utilizing that knowledge. Mostly this showed up in their principles of DOD, or among the questions they ask themselves when applying DOD in practice. Dale Kim exemplified such an understanding of the values you are working with using approximations to trigonometric functions.

> **Dale Kim** – For example, a common optimization that occurs is looking at a function that is heavy on trig function evaluations. Most of the time, you have no control over the implementations of sin() or cos(); the only alternative is to write your own implementation. People are not likely to do this in order to improve the performance of this function. However, it may be the case that the vast majority of angles being passed into sin() and cos() are close to zero. If that's the case, you can approximate sin(x) by just returning x and replace cos(x) with 1. This is not always a valid optimization, but you may find this to be suitable for the problem if you know the constraints and the frequency of the data values.

Utilizing knowledge of data does not just apply to optimizing sub routines though, but also to figure out where you should best utilize your efforts, as noted by Marc Costa and Tony Albrecht.

> **Marc Costa** – ... you should not only focus on the data transformations, but also on how common they are. That's known as the Pareto Principle or 80/20 rule: you want to focus on the code that runs 80% of the time, since optimizing the 20% doesn't have a Return On Investment as big as optimizing the 80%.

> **Tony Albrecht** – The frequency and patterns of the input and output data is an important part of designing your code around your data. Understanding when you're dealing with large sets of data, or small edge cases, can help you focus where you need to be best be applying DoD.

While all participants supported the notion of taking advantage of access and pattern knowledge of the data, there was more variation in relation to domain specific knowledge. Dale Kim, Richard Fabian, and Marc Costa argued that taking advantage of domain knowledge was a key part of DOD, Balázs Török, however, was a bit more restricted.

> **Dale Kim –** Anything that you can use to your advantage to solve your problem should be utilized. That is the whole point of data oriented design, use all the data you have about the problem, your knowledge of it, the hardware, everything.
>
> **Richard Fabian –** DoD is about using what you know, the expectations you have, and programming while considering them. For example, you cannot develop a lossy compression algorithm without DoD, as without considering the qualities of the data, then there can be no expected bias in the values and thus no "likely" sequences. DoD asks us to consider what we know, and help the computer by providing guidance, and not hiding information from it.
>
> **Marc Costa –** Domain specific knowledge is definitely taking advantage of DOD, e.g. when applying a gaussian filter to an image, you can apply a separable filter instead of the full filter for memory access and run-time performance gains.
>
> **Balázs Török –** If it [domain specific knowledge] means knowledge about which cases are more likely then definitely, but if it means applying some algorithmic optimizations that I don't consider part of DoD.

An issue here is that domain specific knowledge can have different interpretations as Balázs Török highlighted. However, my interpretation of the other answers is that they also consider the use of domain specific knowledge in the extended sense.

Explicitness About Knowledge

My understanding of the previous sections and the answers is that it is not enough to only have knowledge about different aspects of the problem you are solving, but also to be explicit about acting on that knowledge. As previously noted by Stoyan Nikolov the designer should embrace the knowledge of the platform, rather than hiding it. The simplicity aspect mentioned by Tony Albrecht in my view also clearly suggest being explicit about knowledge and acting on it, as that is required for an efficient simple solution. Richard Fabian also brings up how explicitly acting on our knowledge is the only way to reach an optimal solution.

> **Richard Fabian –** Using our knowledge also applies in not doing work at all. Consider the simple idea of not playing very distant or quiet sounds when nearby or louder ones are playing. Without putting that idea into the code somehow, there is no way a compiler could happily decide to do that for you and be sure it is doing the right thing.
>
> There are many many examples of this kind of thing, but people often call them premature optimisations because the act of thinking about cost seems to have been given a bad name.

This explicitness is particularly evident in many of the answers to the request for elaboration on what participants considered indicators of poor application of DOD. Aspects such as virtual functions, branch heavy code, and random memory access patterns were brought up by multiple participants. These aspects were sometimes mentioned in both the academic and industry literature, but rarely were any deeper reason for avoiding them or alternatives provided. The line of thinking behind avoiding these aspects was much clearer in the elaboration given by some of the participants.

> **Dale Kim** – ... virtual functions and abstract interfaces implies that you're going to call some code but you don't know which code you will actually call. Very rarely is it the case that you don't know what code will execute. Any time there's virtual functions, you can almost always replace it with sorted state that executes a concrete function on the whole group at the same time instead of one heterogenous array of data which calls into random functions every time.

> **Balázs Török** – ... [Many divergent code paths and special cases in the most frequently called functions] shows that the author didn't know the data and didn't consider the instruction cache as something important. In many cases having the data sorted or bucketed by those conditions that would cause the divergence allows to have very tight loops and good cache efficiency.

> **Richard Fabian** – When you write DoD, you do not tend to write things which operate by magic (that is, not by virtual calls, or callbacks, or hierarchical fallthroughs or tree traversals, or via calls through to script which call back to code.) DoD tends to do things one chunk of work at a time. Each of these chunks are usually easier to profile, but they are also easier to reason about when it comes to the data they depend upon.

Entire Pipeline

Thinking of data transformations throughout the entire game development pipeline was mentioned by some of the industry literature, and was also present in some of the interviews. This notion is brought up both by Marc Costa and Richard Fabian.

> **Marc Costa** – DOD well applied is a holistic approach, meaning that the mindset needs to be applied to the entire pipeline of transformations, not just to the one right in front of us. If pre-computing some results when baking data (game editor data representation to game engine) means we can avoid a millisecond of work at run-time (and we can afford the extra memory usage), there's no point in micro-optimizing the transformation to take half a millisecond. That's still half a millisecond too much compared to the alternative.

> **Richard Fabian** – If you are looking at how you are loading data in your application, you might find there are ways in which you always do some things together. In games you might find you always load a character data file, then always load the meshes, sounds, animations, textures, and any AI scripts. The way the engine is set up, this might not be a direct link, so you can use your knowledge to decide to collect assets related not by type, but by knowing they will be loaded all at the same time and build them into a bundle of data all loaded at the same time, then handed out to each system on demand but from memory. The way a lot of PCs work, they still have spinning disks, so cutting the number of individual files per character down can save quite a bit of time, as opening and reading even the smallest of files will take as long as it takes for the head of the disk to have a chance to read the data. Remember, even the very fastest spinning disks (10k)

> take 6milliseconds per revolution of the disk, so on average it will take 3 milliseconds to open your file. If you have 10,000 files to load for your 100 characters with their 100 different assets each, then you are going to spend around 30 seconds just waiting to open the files. If you lump each character into little zip (not even compressed) with all their assets, then you can expect it to take 100th the time on open file.

Summary

My interpretation of the participants collective understanding of DOD is that knowledge of hardware in a wider sense is considered important, as is knowing qualities of the data you are working with. The wider context you are working in, and the flow of data through your systems should also be taken into account. It might be that the problem you are trying to solve could be better solved earlier in the pipeline, or that you can reuse results from a previous calculation. All this knowledge should be explicitly used to create a simple, direct, and fitting solution to your given problem.

5.2.3 Where to apply DOD?

The participants were also asked what types of problems were fit or unfit for a DOD approach. As with the understanding of DOD there were different answers to these questions. All the participants took a quite pragmatic approach to answering these questions. From my understanding, both Dale Kim and Marc Costa argues that a DOD mindset can and should be applied to all cases.

> **Dale Kim –** I believe it can be applied to all problems. My experience thus far has shown that it's beneficial in every single place that I've attempted it. The main problem is that some problem domains have very well known implementation styles and it may be difficult to overcome the hump of solving those problems in a different way. Usually, this falls away once you truly understand the problem.

> **Marc Costa –** DOD as a mindset should always be applied, what varies, though, is the degree you can apply it. There's only so much you can accomplish with loops over arrays of inputs and outputs and, as software grows, complexity arises. For example, if you want to let users of your software add features without modifying the source code, you'll need to implement an extensibility mechanism into it, probably a plug-in system. A plug-in system is inherently not very data friendly, but it can be designed in a way that the operations it allows the plugins to do, work in a DOD fashion, e.g. adding an API to integrate tasks into the job scheduler of the application.

The other participants seemed a bit more reserved in when it was suitable to apply the paradigm. A common thread was how much data was supposed to be processed and whether or not performance was an important consideration.

> **Tony Albrecht –** Anything that uses data. The more data you use, the more you should be applying DoD principles, particularly if performance is a consideration (which it should be).

> **Balázs Török –** Basically, any problem where high performance is crucial and a lot of

data needs to be transformed, especially real time applications.

Stefan Reinalter – Transforming large sets of data, e.g. many of the operations that need to be performed in a modern video game running at 60 frames per second.

DoD should be applied when the benefits clearly outweigh the disadvantages.

Richard Fabian – When you care about producing something within the constraints of possibility, where you are likely to hit hardware and complexity limits, such as working on a non-trivial software project.

Stoyan Nikolov – Any software problem that transforms a non-trivial amount of data. There are industries where this is obviously the case – video games, financial, ML, data processing etc.

When asked directly where DOD should not be applied, participants brought up aspects such as performance not being a concern, or other external pragmatic reasons. Particularly interesting are the aspects noted by Balázs Török and Tony Albrecht, that technically, if you do not have information about your data, you cannot really use or create a DOD solution.

Stefan Reinalter – If OOP leads to less code and is probably equally fast (or the performance impact really is negligible), it doesn't make sense to apply DoD to everything.

Richard Fabian – Prototypes, demos, toys, anything where you can throw overly powerful machines at the problem without cost to you, such as experiments where you can leave a machine running a poorly written python script overnight because developing a better solution in C++ which might run in ten minutes would take more than 24 hours to write, and there's a night's rest if you don't.

Stoyan Nikolov – Many problems are far heavier on logic than data itself - in those cases other techniques might be easier to apply. For instance, a system that manages plug-ins within software package. It is unlikely that there would be more than a dozen (hundred?) plugins – but the logic of loading, unloading, updating them etc. could be quite complicated. Such a system I'd design in a very traditional OOP style – I see no benefit in building a data pipeline when I have no good knowledge of the data – the plugins are literally "black boxes". When you have black boxes – OOP is quite good.

Tony Albrecht – You can solve any problem with a DoD mindset, but it may not be the simplest solution, initially. If you don't understand the data being used, you can't really use a DoD solution – simple procedural or OO or functional implementations can then help you to better understand your problem and the data being used – then DoD can be used. In the same way you can write an OO implementation to do anything, you can do the same for DoD

Balázs Török – Problems where having a solution ready quickly is more important than having it perform well, or ones where the data or the underlying hardware are poorly specified.

Summary

My understanding of the participants answers is that DOD can always be applied, but that it might not be beneficial to do so in situations where performance is not a concern, or where there are uncertainties or poor understanding of the underlying hardware or data.

5.2.4 Benefits & Drawbacks

The participants were also asked a series of questions relating to benefits and drawbacks of applying a DOD mindset. Participants were first asked a opened ended question of what they considered to be the main benefit of applying a DOD mindset, before more specific questions related to productivity, bug fixing, and interfacing with libraries was presented. The following paragraphs are a collection of the topics that were raised, as some of the participants touched on the different topics in different questions.

Maintenance, Bugs, & Testing

Easier maintenance in general was brought up by multiple of the practitioners. Stoyan Nikolov notes that DOD promotes maintenance by forcing the programmer to think about the evolution of a system. Dale Kim states that improving performance, which he argues is part of maintenance, is so much easier that if nothing else changed, DOD would still be worth it. Tony Albrecht argues that DOD code is easier to maintain because of the focus on simplifying your problems.

> **Stoyan Nikolov –** I also think it [DOD] simplifies the maintenance and extension of systems. It pushes designers to think hard about the evolution of a system when requirements change. With traditional OOP it's easy to do a "quick patch" on a design that might solve a short-term problem but become a long-term liability.
>
> **Dale Kim –** Improving performance (which is a huge part of my job) is so much simpler that if you made other kinds of bug fixing the same, the benefit is so massive that it would still be worth doing. For example, improving performance in typical object oriented code is nearly impossible because the data arrangement and dependencies were made at too granular of a level (if at all, often that code grows organically to the point where any data design you started with was abandoned long ago). You can only do excruciatingly detailed micro optimizations which will only net you maybe 20% improvement.
>
> **Tony Albrecht –** The main reason for DoD, in my opinion, is to simplify your problem as much as possible, so that you are doing simple things to simple data. It produces code which is easier to maintain (as it is simple), is easier to optimise (inherently cache friendly) and is easier to parallelise (data coherency).

Related to maintenance is the topic of bugs. Two questions presented to the practitioners asked them if they had experienced some types of bugs occurring more or less frequently. The answers here varied a bit, and two participants chose not to answer the question. Stoyan Nikolov brought up that many typical C++ issues started to appear less frequently, similarly Stefan Reinalter brought up the topic of state related bugs.

> **Stefan Reinalter –** Less worrying about “is there a data race somewhere?”„ “does this function touch global state?” and “who changed that instance’s data?”.
>
> **Stoyan Nikolov –** Typical C++ memory and object-related issues were far fewer: null-pointer derefs, leaking references of smart pointers (and thus leaking memory or resources), wrong implemented boilerplate methods like copy & move ctors/operators.

None of the practitioners came up with any types of bugs they would particularly attribute to the use of DOD, but Marc Costa noted that some issues such as indirect indexing into 2 dimensional arrays might appear more often due to being closer to the data. Others brought up that fixing bugs in general became simpler because the code was easier to reason about or debug.

> **Marc Costa –** I haven’t seen bugs specific to DOD start to occur, however some of the bugs I’ve always made regardless of design choice are still there: wrong loop ranges, wrong mathematical operations, wrong mapping between data formats (e.g. linear 2D array to tiled 2D array). These kind of bugs might appear more often in DOD scenarios since you’re usually closer to the data representation, however they’re usually quite easy to spot and fix.
>
> **Richard Fabian –** Unexpected state bugs didn’t seem to occur any less frequently, but they were much easier to track to cause and squash. I’d say that the number of bugs per feature stayed about the same, but debugging became easier as it was easier to tell where data was modified.
>
> **Dale Kim –** Generally, it’s just easier to reason about the code and data and easier to make changes that are isolated in nature to actually fix bugs.

Testing

The topic of testing was only brought up by some of the practitioners. However, all of them highlighted testing being an easier activity. In particular this was due to the focus on clear inputs and outputs to functions. Marc Costa even considered it a characteristic they expected to see in projects that apply DOD correctly.

> **Marc Costa –** Tests: DOD makes it easier to test functional parts of the code. Note, this doesn’t imply Test Driven Development
>
> **Dale Kim –** ... testing becomes so much simpler/easier since you know what the data should be in terms of inputs and outputs that writing tests for your code becomes self evident and easy since the code doesn’t have any unnecessary (or minimal) abstraction.
>
> **Stefan Reinalter –** Testing is a million times easier when you have pure functions. Feed them input values and you know exactly what the output values are going to be and can test for them. Compare this to OOP code that needs other class instances, global state and mock objects in order to test a single function of a single class.

Productivity & Tooling

Related to bug fixing and maintenance is general productivity. Stoyan Nikolov and Stefan Reinalter both bring up a productivity deterioration when applying DOD to a new problem, but also note benefits to the approach. From my understanding, Marc Costa seem to have experienced the opposite, struggling more with new problems when choosing an object-oriented approach. Richard Fabian notes from my understanding, a general increase in ability to write new features.

> **Stoyan Nikolov** – Creating new systems requires more design time and thought – I feel it's slower than if one would use just OOP. This is compensated by making it easier to add features to existing systems, debugging them, and reducing significantly performance regressions. Especially in performance-critical software, regressions are a real issue and can be very hard to track – with DoD this was significantly improved.

> **Stefan Reinalter** – Writing unit tests was more fun, I had to spend less time on optimization (because code written with DoD in mind tends to be faster). However, I sometimes get the subjective feeling that I get less work done in the same amount of time when trying to apply DoD to a new problem (compared to a OOP solution).

> **Marc Costa** – Another advantage is avoiding *analysis paralysis*. This refers to the inability to make decisions when starting to write code to solve a problem due to the large space of possible solution vectors. I had this problem when trying to solve problems with OOD: what classes should I use, what should go into each class, is there a design pattern that applies here? I would also start writing classes and mock up interfaces before I wrote any code that solved the actual problem.

> **Richard Fabian** – When I program in DoD I tend to spend around 60-80% of my time writing features, with the remaining 20-40% split evenly between debugging and maintenance. When programming OOP, due to the difficulty in figuring out where state is changed, who changed it, when, and what state things are in, I find that maintenance and debugging take up 60-80% of my time. New features are desired, but maintaining existing code dominates the development. New code always seems to exercise code paths not usually travelled and in there we often find bugs. The DoD way tends to have simple linear transformations which either work, or don't. Introducing new code doesn't affect it in the same way.

The lack of good tooling support was also brought up as possible productivity deteriorations by both Marc Costa and Stefan Reinalter.

> **Marc Costa** – A possible productivity deterioration is debugging certain code snippets. Debuggers are basically state driven, I.e. when breaking into the debugger, you can inspect the current state of the program. However, due to how some memory ends up being laid out or organized, it might be hard to extract meaningful state from what the debugger gives you. Custom debugger visualizations help, but current debuggers don't make it very easy to use them.

> **Stefan Reinalter** – DoD can make it harder to debug/diagnose certain classes of errors due to the way data may be spread into different "chunks" or classes, e.g. when considering a pure DoD-Entity-Component-System, where you no longer have a single

instance of a class that can be viewed in the debugger.

Complexity

Simplicity was brought up by many of the participants in their understanding of DOD, it should therefore come to no surprise that many of the participants consider DOD code to be more simple than object-oriented programming. Many of the participants did not have much experience with functional programming, making it hard to do any comparisons to that paradigm. It was also mentioned that while the code might end up simpler when following a DOD mindset, the process of getting there might not be simpler than when using other paradigms.

Dale Kim – [The code is considered] Drastically simpler than OOP. Hard to compare with functional programming since I haven't written anything as extensive in a functional language, but my perception is that for me to achieve the same quality level in a functional language would require far more work.

Marc Costa – I find the code to be generally simpler. By focusing on the transformations alone, the code usually ends up in a straightforward layout, especially when compared to the complexity even basic OOP code adds. Of course, there are still sources of complexity, like the problem itself (inherent complexity) or the plugin system I described in #8, but even these cases benefit from simpler programming building blocks than what OOP gives you.

Stefan Reinalter – Depends. One definitely has to think more about the data layout upfront, but that saves time in the long run, like not having to do many of the optimization tasks that are typically done at the end of production.

When applied correctly, the code should not be more complex than the OOP counterpart – quite the opposite.

Tony Albrecht – In a lot of ways the DoD code is simpler than OOP (I've not done a lot of FP). One of the issues with OOP is the evolution of inheritance trees – they just don't scale well over time. DoD solutions scale better – you can add in new data or functionality without adversely affecting the existing data or functionality.

Richard Fabian – No, it's definitely less complex than OOP, but possibly more complex than FP, as DoD does still maintain a large amount of state compared to the pure functional languages.

Stoyan Nikolov – No – just the opposite, I believe it's simpler. The design phase of it was more complex though.

Balázs Török – It might certainly look more complex to people who are not used to this paradigm but that's probably true the other way around as well.

Interfacing with other libraries

As DOD is not the dominant approach taken when developing most software I thought it would be interesting to get the practitioners experience in relation to interfacing with existing libraries. From my understanding the main concerns are poor performance, and the issue of having to marshal data back and forth between your own layout and the layout that the library requires, which can lead to poor performance and increased complexity. To a certain extent this is also related to customizeability. For example, as noted by Marc Costa memory management is often handled by the library, rather than being something the user can control.

> **Marc Costa** – There are many sorts of friction between code that follows DOD and code that is OOD:
>
> - Memory management: most of the time, memory lifetime for the library is handled internally, tied to objects and generally they lack the possibility to specify what memory to use (instead of allocating from the standard library heap)
> - Complex interactions between objects: OOD based libraries will use different objects to represent the various concepts in the library, which interact in arbitrary ways. Managing the objects then becomes the responsibility of the client.
>
> **Dale Kim** – Being stuck with terrible performance and incomprehensible code. There are some very popular GUI libraries which have an insane amount of object orientedness that it's very difficult to understand what the structures are and how they actually map to commands the GPU will ultimately interpret.
>
> There are also problems at the interface layer where you may need to marshall data back and forth from your format to the libraries format, but this problem is not specific to non-DoD libraries. All libraries suffer from this since you won't always have your data in the same format as the library expects.
>
> **Stoyan Nikolov** – Adapting the data can be challenging and lead to reduced performance and clarity, because it requires this "adaptation" phase. The results of such libraries must be plugged back in the main system and get re-organized.
>
> I can imagine that software that heavily relies in a core part to such an external library might have serious problems. The work of adapting the data (and complexity) might out-weigh the "real" work of the library/system.

Other Aspects

Other benefits outside of the ones above were also mentioned. Richard Fabian brought up the notion of not overstepping budgets of all kind. When asked to elaborate on what types of budgets he meant he indicated both product runtime budgets and development related budgets.

> **Richard Fabian** – Memory budgets (for CPU/main mem and also for GPU), loading time budgets (you will have real hardware on which the application must run, and that will have an IO throughput, so you will have a budget even if you don't know it), network bandwidth budget and error tolerance too, anything which can get too big (save game size, crash dump size, UserGeneratedContent size), or run out (file handle count, active network connections). Then there are other budgets pertaining to the development such as how long you're willing to let people develop the game without telling them if they are inside these budgets (if you don't know how long you're willing to do that, you're

> effectively telling them they will be cutting and redoing work at some point, but you won't tell them when.)

Summary

From my understanding of the answers above working with a DOD mindset results in simpler code that is easier to maintain and optimize. However, a longer design phase might be required when working with a DOD mindset than other paradigms. Due to the focus on inputs and outputs to functions, testing and bug fixing becomes easier, although the frequency of bugs might not be reduced. Certain practitioners reported a productivity decrease when working with new problems, while some reported a productivity increase. Related to this is the issue of tooling. Since DOD is not a dominant way to develop software there are not many tools that support a DOD workflow. Interfacing with libraries following different paradigms seemed to be mainly a concern from performance and complexity viewpoints. Lastly, as noted by Richard Fabian, whether you know it or not, you as a programmer have budgets, and DOD can help you stay within them.

5.2.5 Hurdles of Learning DOD

It was noted multiple times in the literature chapter that working in a DOD manner could be challenging because programmers are not used to it. I therefore included a question asking participants what proved to be the biggest hurdle to overcome when they started learning DOD.

The most common hurdle mentioned by the different participants was moving away from an OOP mindset, which, from my understanding, was the only or main tool some of these participants had available. As noted by Marc Costa learning an entire new way of thinking from scratch is non trivial.

> **Marc Costa –** The biggest hurdle was letting go of OOD mindset. I had been trained on OOD for years before I even wrote one line of code on a professional setting. By then, OOD is all I knew and the only way I knew how to approach and talk about problems, so everything needed to be solved in terms of classes, objects, instances, and design patterns.
>
> This is a big deal, since OOD encompasses the entirety of your toolset, so you have to start learning a new toolset from scratch. That's a daunting endeavor.

Richard Fabian brought up the issue of peers criticizing him for taking the "wrong" approach, stating that other people, were the biggest hurdle to overcome.

> **Richard Fabian –** Other people. Many people said I was coding wrong, as if OO was somehow a universal right way to do it.

This aspect was mentioned by Richard Fabian already in the first question about how long he had been working with a DOD mindset.

> **Richard Fabian –** ... I've always been thinking DOD, except for a time between 2001 and 2006 where I was an OO zealot due to overwhelming support and books suggesting

> it was the only good way to program software of any appreciable size. Books like the Gang of Four Design Patterns book and Code Complete were seen as important and true, and I was not experienced enough to criticise them at the time.

Dale Kim notes that the biggest hurdle for him was how to actually work in a DOD manner. He also brought up the related issue of how different programming languages' design also impact the approach and choices a programmer makes.

> **Dale Kim –** How to actually do it [DOD] for real. The way it is described seems *magical* if you have never done it before; people claim what seems to be unreal performance gains and also purport to have large maintenance benefits. If it's so great, why doesn't everyone else do it?
>
> The main thing is that many people think of programming in terms of what the programming language of choice lets them or makes easiest to do. This has almost no bearing on what the hardware actually wants to do. For the first few years after college, I thought I was doing "Data oriented design" because I was trying to lay out my data to account for the cache, but I was hamstrung by writing all of my code within the confines of what the C++ language design wants you to do (think writing classes with a bunch of different data members and trying to move members around based on which members are used in different functions. It simply doesn't scale).
>
> This is data oriented design in the smallest and most basic sense because you're trying to be aware of what the hardware likes, but the benefits are so small due to trying to shove it into a language design that isn't conducive to the hardware design.
>
> It's far simpler to look a the hardware in isolation and figure out how the data layout and flow should be for a problem. Once you figure that out, write the bare minimum code to do exactly that in whatever language of choice you have. Ignore what the language tries to make you do; implement exactly what is most beneficial and direct for the hardware to execute.

Summary

While this study has quite a small number of participants, it was interesting to see that so many struggled to get out of the OOP mindset in order to learn something else. My personal interpretation of this apparent struggle is that we all should be careful promoting any sort of paradigm as the "correct" way to approach programming. OOP was at one time seen as the dominant approach to software development, but now functional programming seems to be becoming increasingly popular (again). My personal belief is that there is no "one true way" to develop software. Both because different problems require different solutions, and because people think in different ways. Promoting an approach as the "one true way" leads to fitting a problem to a solution, while also limiting creative problem solving. This also somewhat relates to the point made by Dale Kim, as the design of a language encourages if not forces certain approaches to solving problems.

5.2.6 Misconceptions

Based on the discussion surrounding the more holistic view held by the industry, I also included a question regarding common misconceptions about DOD. The belief that DOD is all about cache misses and low level programming was raised by some as a common misconception. While others brought up the misconception that DOD was less maintainable, takes more effort, or is not needed anymore.

> **Richard Fabian –** It's all about cache-misses. DoD code is all handwritten assembly and it's unmaintainable. DoD is just applicable to old PS3 and XBOXes. DoD isn't for gameplay code.

An interesting note is brought in by Dale Kim who takes DOD out of a pure performance focus.

> **Dale Kim –** Say that your primary concern isn't performance care about some other metric, like bug maintenance time. A data oriented approach would be to figure out what your main constraints are to solving your problem and how it affects bug maintenance time.
>
> You should gather data about where all the bug fixing effort goes and study why it takes so much time for those areas of the code to be maintained. You need to think very critically about why things are costly; it's very common for people to effectively sweep things under the rug by just claiming the problem is "hard" without actually figuring out why so much development effort is put into that code. Ultimately, you may come up with an approach where the final solution isn't the highest performing implementation but is far lower maintenance cost.

Stoyan Nikolov brought up another interesting point, that DOD does not focus on giving answers, but on asking the right questions.

> **Stoyan Nikolov –** DoD is much less of programming paradigm than OOP (which has established and accepted rules) but more of a way of thinking. Some concepts in DoD seem even trivial to developers, but when trying to effectively use them, they see the complexity required to get good results.
>
> Many developers expect from DoD complete answers, however DoD is more about asking the right questions.

Summary

The misconceptions listed by the participants is in line with my thinking that there is more to DOD than just cache optimization. Dale Kim's comment about a non-performance focused DOD is interesting as it somewhat portrays DOD as a way to solve problems. This is somewhat in line with what Stoyan Nikolov states in that DOD is more about asking questions, rather than having a set of answers.

5.2.7 Origin

I included the question regarding the origin of DOD as I could not find a definitive answer during my literature analysis. Tony Albrecht notes that it was discovered due to necessity, by multiple people around 2007, with it being named in 2009. The point about it being discovered by necessity was also brought up by other participants, in particular it was important on the PlayStation 3 and Xbox 360 platforms. There are some diverging opinion on the roots further back in time though. Marc Costa notes that the common theme of the early DOD publications was better hardware usage, and that that in and of itself is not new. He notes the works of Michael Abrash[61] as an example of such thinking. Stefan Reinalter draws a link to the C language being the origin of DOD, similarly Stoyan Nikolov draws a link to procedural programming as the roots of DOD. Richard Fabian draws the link back to super computers and vector machines, but notes that the name did not come about until the PlayStation 3 and Xbox 360 generation.

> **Tony Albrecht** – It was discovered by multiple people around the same time. It was needed as our CPUs were getting faster, but the memory speed wasn't increasing at the same rate. With memory speed as the bottleneck, we had to think about optimisation and program design in a different way. Noel Llopis, Mike Acton, myself were discussing the principles of DoD in around 2007 (and probably a few others too – its been awhile. Christer Ericson used DoD principles but never really broadcast that publicly). We all discussed, blogged and spoke publicly about how we were thinking about code but the name didn't stick until I think Noel blogged about Data Oriented Design. My talk in 2009 – Pitfalls of Object Oriented Programming – was a catalyst that helped push data as a primary consideration in code design, but there were many of us working to solve the same problems in similar ways.
>
> **Stefan Reinalter** – At least from a paradigm point of view, people programming in C never had the problems we often face in big (old) C++ OOP codebases, because the language itself doesn't offer the facilities that make it so easy to corner yourself into overengineered abstractions. C doesn't offer inheritance, virtual functions, etc.
>
> If you tell a C veteran about your newfound way of doing large transformations of data using pure functions, supplying all the input and output data without touching mutable (object/instance) state, he'd probably look at you and ask you "Of course, why would you do it any different?"
>
> **Stoyan Nikolov** – My experience in game development dictates that the quest for better performance is at the root for "naming" DoD. The principles however are far older and rooted in procedural programming. I think that giving a name "DoD" was fundamental to making this style of software design more popular. People now have a common lingo and can describe common ideas under an umbrella term.
>
> **Richard Fabian** – Super computer technology and vector machines were the real start of it, but the origin of Data-oriented design as a term of note was the dire need of an alternative to OO programming on the in-order processors of the PS3 and Xbox 360. People united around a need to understand hardware and data, so from that point of view, I consider it to be the origin of it gaining sufficient identifiability to named.

Summary

While some more light has been shed on the origin of DOD I unfortunately did not get the direct answer I hoped for. My current understanding is that DOD might have roots further back in time and fields such as procedural programming and super computers, but that the name and the incarnation we see now is rooted in the video game industry. Due to the scope of the thesis I did not further pursue the works by Michael Abrash[61], nor the topic of super computers, vector machines, or the history and design of the C language.

5.3 Limitations

The conducted interviews and analysis were conducted in a qualitative manner, and is built on my interpretations. Additionally, the participants were selected from a small pool of practitioners who had publicly discussed the topic. Due to the small number of participants, and the potential for biases in the questions as well as the practitioners' backgrounds, I chose not to do an quantitative analysis as the one I did in the literature chapter. Rather, I treat this chapter as a way to highlight potential future research endeavours.

5.4 Summary of Interviews

This chapter has given an overview of various interviewed industry practitioners perspective on the topic of DOD. My interpretation of their collective understanding is that when working with a DOD mindset you are supposed to make explicit use of everything you know regarding the context you are working within and the data you are working with, in order to create a simple, direct, and fitting solution to a given problem. The line of thinking behind DOD can be employed anywhere, but it might not be beneficial in situations where performance is not a concern, or there is a poor understanding of the context you are working within.

According to the interviewees there are multiple benefits to working in a DOD manner, such as simpler code, and easier performance maintenance, bug fixing, and testing. However, a longer design phase might be required for the above benefits. While some participants reported productivity deterioration when working on new problems, others reported a productivity improvement.

From my understanding there is currently also a lack of proper tools to support a DOD workflow, an example is debuggers that was developed with other workflows in mind. Interfacing with existing libraries was considered a concern mainly from performance and complexity viewpoints.

The most common hurdle for learning DOD was to move away from an existing OOP mindset, which seemed to be the practitioners' main or only tool. My take from this was that one should be careful about promoting any sort of paradigm as the "one true solution".

When asked about common misconceptions regarding DOD the participants commented on the belief that DOD is only about cache misses and that it is not maintainable. Additionally, the participants raised interesting points regarding how you can apply DOD in a non-performance setting, and that DOD is more about asking the correct questions, rather than providing set answers.

On the point of DOD's origin, my understanding is that DOD has several roots to earlier paradigms and hardware focus, but that the DOD we see today was born in the video games industry and promoted around 2007, gaining its name in 2009.

5.5 Conclusion on Research Question 2

Research question 2 asks "What are the core characteristics of DOD?". In order to answer this question, I decided on a limited list of characteristics, three to four items. I developed an initial list of suggested core characteristics based on direct and implied references to central aspects of DOD as observed in the literature, focusing on the problem solving aspect of DOD. I further refined the list based on the understanding gained from the specific questions posed in the interviews. In the end, I identify the following as the main four characteristics of DOD.

1. Focus on solving your problem at hand, rather than a generic one

The first core characteristic is the focus on not generalizing the problem you are currently solving. Generalizing a problem can lead to sub optimal solutions for your specific case, and might lead you to deal with edge cases that do not exist within your specific problem space.

2. All kinds of data should be considered

The second core characteristic is that all kinds of data should be considered. Data is everything from understanding of the problem domain, to knowledge of the target platform, or patterns within the expected values. This is further discussed in Section 6.2.

3. Decisions should be made based on data

The third core characteristic is that decisions regarding the implementation of a solution should be based on data surrounding the problem at hand. The data should guide the implementation, rather than some abstract mapping of the problem domain.

4. Focus on performance in a wide sense

The fourth core characteristic is the focus on performance in a wider sense. Performance is not just about short frame times or less memory consumption. Rather it depends on the goal and problem at hand. As noted by Dale Kim, it might be that you care more about bug maintenance time, good performance in that situation could be the time it takes from a bug is discovered until it is fixed.

Other Characteristics

Examples of other main characteristics that have been evaluated and left out are hardware knowledge, and the various patterns that are often seen in DOD solutions (relational programming, SoA, etc). The reason for not including the various patterns is that DOD is about solving problems based on your actual data. Applying patterns without a critical look at your data would be in opposition to the first characteristic. In my view, hardware is yet another piece of data in DOD, as conceptualized above. However, the specifics and extent of the analysis depend on the problem at hand. Hardware is an important piece of data when discussing DOD in a run-time performance view, but might be less important in a security perspective. Therefore a generalized approach to analyzing hardware in the context of DOD could rather be seen as a limitation of the description of DOD itself. This is specifically alluded to in the answer given by Dale Kim in Section 5.2.6, and also how I understand Mike Acton's view as noted in Section 4.5.9. The benefits of applying DOD (such as maintenance benefits and easier testing) is not considered as core characteristics because they are a result of a DOD approach.

6 Initial Understanding of the Application of DOD

Research question 3 asks "How does one approach software development with a DOD mindset?". This chapter presents my initial understanding of the application of DOD as a result of the literature review and industry practitioner interviews.

6.1 DOD Visualization

From my understanding, data-oriented design is more a loose process/methodology, a way of thinking, rather than a set of strict steps. Figure 6 is a visualization of how I understand DOD. It is important to emphasize that neither the figure, nor the following description is meant as a step by step guide (even though it appears as such) to "do" data-oriented design. Rather it is a way to exemplify how one could approach a problem in a data-oriented manner.

You would start by gathering as much information on your problem and context as possible. What and how to gather that information is covered in Section 6.2 and Section 6.3. This data should be analyzed and used to create a simple, fitting, and explicit designed solution to a given problem, which is then implemented. The implementation will most likely lead to new insights i.e. data, which again needs to be analyzed in the context of the already existing data to create an even better solution. This process is iterative, and should be carried out until a satisfying solution has been found.

6.2 What is data?

It might be tempting to think of the term data in data-oriented design, as simply being bytes and the values those bytes represent. However, while including those aspects, the term encompasses a wider range of aspects. Data is information about the hardware you are working on, patterns and expected

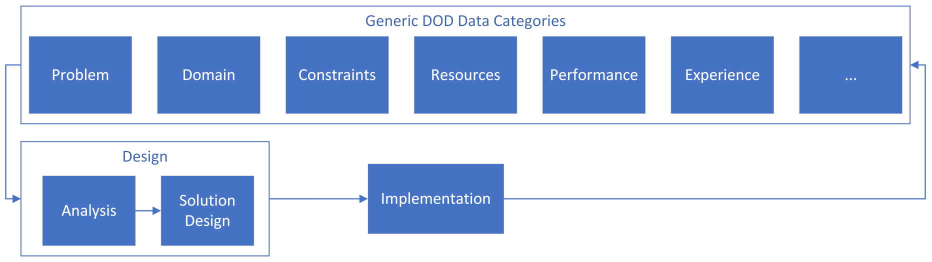


Figure 6: A Visualization of the DOD process/methodology

values you are working with, data flow throughout your systems, and details of the problem you are solving. Figure 6 shows a set of generic data categories that covers what I view as data in DOD. However, this set is not necessarily complete. The ellipsis is there to allow more categories to enter the model. Additionally, the various categories might not always be present, or completely disjoint, as that depends on the situation and problem at hand. For instance, performance data will not be present until a solution is actually in place. The following paragraphs give a brief explanation of the different categories.

Problem

Problem data is data gained from a situational analysis and a high level analysis of the problem you are attempting to solve. For instance, what is specific to your current situation, or are there any specific requirements that needs to be kept in mind? What are the steps involved in solving your specific problem, and how can they be divided into sub steps.

Domain

Domain data is knowledge of the domain that you are working within. For example, if you are working with a standard file format, does all the parts of that standard need to be supported? Is it possible to only implement a subset of the standard because the other parts are rarely, if ever, used?

Resources & Constraints

Resources and constraints are to a certain extent two sides of the same coin. They both relate to the context which you are solving your problem within. Example of data here are limitations of the target hardware, or memory budgets and deadlines.

Performance

Performance data is data about the performance characteristics of the current solution, given that one exists. How fast is the current solution? How much memory does it use? How far is it from a defined optimal solution?

Experience

Experience data relates to the programmers previous experience solving problems related to the current one. This experience should be built upon real data, rather than just being a hunch.

6.3 How do you gather the data?

Gathering the various data can be done in different ways. The generic categories points to certain places where you can obtain data. Acton[38] shows that simply printing out values and storing them in spreadsheet for analysis is a great way to find patterns within those values. The interview participants in the previous section were asked what questions they often ask themselves when working in a DOD manner. Some of these questions are listed below.

- **Richard Fabian –** What is the minimum amount of data required to solve this problem?
- **Tony Albrecht –** What is the data input, what is the output, how do I transform the input to produce the output in the simplest way possible?
- **Marc Costa** – Do I need to do this? (can I just do nothing and get the same result from somewhere else? E.g. previous transformations)
- **Marc Costa –** Can this be done earlier? Some transformations in a game engine can be avoided if e.g. the asset pipeline generates the data beforehand (need to balance memory usage & runtime cost)

6.4 Analysis, Design & Implementation

Once the data has been gathered it should be analyzed in context, and used to guide the design and implementation. The design and resulting implementation should be direct, non-generic, and minimalist, i.e. not do more than what is needed to solve the given problem. Abstractions can be used if the problem is complicated enough to require it. However, as noted by Acton[62] these abstractions should be utilitarian and transparent in nature. This means that the abstractions should not hide what transformations are done to the data, nor should they be used to embed the problem domain in the code.

From my understanding, the chair example given by Acton[19] is an example of an abstraction that both hides data transformations and embeds the problem domain into the code. Acton[62] uses a light switch as a metaphorical example of an utilitarian abstraction, stating that nothing is stopping him from just removing the light switch and manually do the wiring himself. In my understanding, I would say that C++ 's std::lock_guard[63] is an example of utilitarian abstraction. The std::lock_guard is a convenience wrapper object for the act of acquiring (or adopting) a mutex and releasing it at the end of the scope. Its implementation is trivial, and using it is optional. No one is stopping you from simply calling mutex.lock() at the beginning of scope, and mutex.unlock() manually at the end of the scope.

In the end, data should be stored by how it is used, and the transformations applied to the data should take advantage of knowledge regarding the data.

6.5 Summary

This chapter has discussed my initial understanding of how to apply a DOD mindset from a theoretical viewpoint. Through the next chapters I attempt to apply this understanding to a case, before discussing the validity of my approach and understanding with industry practitioners.

7 Industrial Case Background

In order to validate the previously presented understanding of DOD I decided to apply it in a real world scenario. My plan was to solve a realistic case, using my understanding of a DOD methodology. Throughout this case I would write a document describing my process. I would present this document to willing industry practitioners in order to get feedback on my approach, and thereby validate that my understanding of DOD is in line with theirs. The following chapters is the same as the process document that I delivered to the industry practitioners, with the exception of some minor differences. These differences were either correction of grammatical errors, formatting, or minor content changes for clarity or error correction. The content changes have been highlighted in footnotes. The Case Description (Chapter 8) as well as the appendix Extended BIM Info (Appendix M) are from an earlier (non-DOD) attempt at solving the problem at hand, in the context of the course IMT4894 Advanced Project Work. However, everything else was achieved within this thesis. The methodology for validation, as well as the practitioners feedback is discussed in Chapter 10.

8 Case Description

This document gives an overview of the problem of streaming architectural models from disc into the VR application VREX.

8.1 VREX

VREX[1] is a Computer-aided design (CAD) Virtual Reality (VR) visualization tool developed by Vixel[2]. This tool allows participants in an Architecture, Engineering, and Construction (AEC) project to meet in a VR version of their project and discuss important issues related to said project. The application does not require VR, as it also allows users to participate through a desktop application. This project is a cooperative endeavor with Vixel to implement a streaming system for Building Information Modelling (BIM) models[3] into their application to help tackle the performance issues that arise when dealing with larger BIM models.

8.2 Current Approach

Currently VREX supports Industry Foundation Classes (IFC)[4] files for importing BIM models. An IFC file is converted into a digital asset exchange (.dae)[5] file through the use of IFCOpenShell's[6] IFC converter. The .dae file is then imported into Unity (the engine VREX is implemented in), where each IFC object ends up as its own Unity game object[7] with a game object parent named according to the IFC model. An illustration of this pipeline can be seen in Figure 7.

It is not uncommon to have multiple IFC files for one building, covering different engineering areas. Importing multiple IFC files in VREX is treated the same way as importing a single file. Each IFC file is converted, creating multiple .dae files which are imported into Unity.

8.2.1 Motivation for a new Approach

There are various reasons Vixel wishes to move to a new approach.

[1]https://vrex.no/

[2]https://www.vixel.no/

[3]The correct name for BIM models is: Building Information Models, which are often abbreviated to BIMs, in this report I will specifically write BIM model, as using both the terms BIM and BIMs can quickly cause confusion

[4]A short explanation of BIM and IFC can be found in Appendix M

[5]https://www.khronos.org/files/collada_spec_1_5.pdf

[6]http://ifcopenshell.org/ifcconvert.html

[7]https://docs.unity3d.com/Manual/GameObjects.html

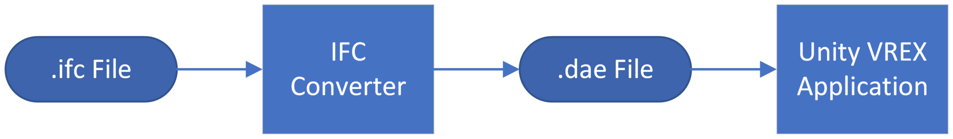


Figure 7: The current IFC import pipeline in VREX

Poor Scalability

The current solution of importing an entire model into Unity does not scale well. Larger or more detailed models will require more resources, even if only a subsection of the model is visible, due to every part of the model being loaded into the application.

Reduced Import and Startup Time

Importing larger .dae files into Unity can be a time consuming process. Similarly, the time needed to launch the application increases with larger models. A streaming solution would incur more runtime logic, but would free up engineer time spent waiting for the asset import and application launch process.

Greater Split Between Application and Model

With the current approach there is a large amount of coupling between the BIM model and the application itself, as each model is effectively its own application. A streaming solution would reduce this coupling, allowing for a single application dealing with multiple models.

8.3 Problem Description

Vixel is looking at streaming of content as a possible solution to the issues mentioned above. For such a solution Vixel has the following requirements.

8.3.1 Vixel Requirements

Functional Requirements

Requirement 1: The Solution Shall Dynamically Load and Unload Objects Based on a User's Position

As implied by the term streaming, the solution needs to be able to load and unload different parts of the IFC model based on the user's position.

Requirement 2: The Solution Shall Support Existing Hiding and Selection Schemes

VREX allows the user to select, highlight, and hide different objects in the world. For example, hiding a wall will reveal the insulation material used within the wall. This functionality must still be implementable in the streaming solution.

Requirement 3: The Solution Shall Support Existing Movement Schemes

VREX allows to user to move around in different ways, such as teleportation or grip based movement. The streaming solution must support the existing movement schemes, and preferably not place any restrictions on future movement solutions.

Requirement 4: The Solution Shall Incorporate a Level Of Detail (LOD) Mechanism

If the IFC model is seen from far away there is no need to load in all the smaller detail objects such as power outlets. Therefore the solution should supply some level of detail mechanism. The mechanism does not need to support low-resolution versions of the models; it is enough not to load the smaller detail objects.

Requirement 5: The Solution Shall Allow For Querying IFC Objects for Their Unique ID to Allow for Extra Information

When conducting architectural inspection the user may need access to extra information provided by the IFC file format. The streaming solution needs a way to map a selected visual object to the corresponding IFC object, such that extra IFC information can be extracted.

Operational Requirements

Requirement 6: The Solution Shall Incur Minimal Overhead

Due to VREX being a VR application keeping a high framerate is essential. Therefore, it is important that the streaming solution has as low overhead as possible. If there is a choice between the user standing in a scene that is not fully loaded and the user experiencing a drop in framerate, then the former approach should be taken.

This requirement is not as clearly defined as the others, as we do not yet know what overheads are realistic, and the models themselves are hard to predict. Vixel wants to spend as much of the frame budget on application functionality, and we will conclude the exact overhead value of this requirement as we are closing in on a solution.

Developmental Requirements

Requirement 7: The Solution Shall be Documented

The streaming solution will need to be maintained by other software developers in the future. Therefore it is important that its functionality and implementation is documented in a design document.

8.3.2 Implementation Outside VREX Codebase

It was decided that this solution would be implemented outside of the existing VREX codebase, to avoid this project being impacted by the ongoing development of the application and vice versa. Additionally, it is considered more important to get a solution that functions well, than avoiding larger rewrites of the system.

9 Developed Case Solution & Process

This section describes the general approach and process taken to develop the solution. This was done in a more iterative and concurrent manner in the actual implementation phase. I will first present the process I attempted to follow, then the data/information that can be extracted from the case and problem approach, before discussing how that data were acted upon to create a fitting solution.

9.1 Process

I attempted to follow my understanding of data-oriented design (DOD). A visualization of that understanding can be seen in Figure 8. The ellipsis in the figure indicate that more categories can be added. As seen from the figure, the general process consists of gathering relevant data (see *generic DOD data categories* in Figure 8) for the problem that is supposed to be solved before analyzing said data. The analysis of the data drives the solution design. Implementation of the solution design will lead to more data, and one should iterate on the whole process until a satisfying solution is found. The figure is meant to be a simplistic guiding visualization. The categories are not exclusive, and they may not exist in all situations.

Figure 9 shows the main data categories I operated with in this specific case, and largely what generic category they map to. The data is further described in Section 9.2. This data was analyzed and acted upon to fuel the solution design, which is discussed in Section 9.3. Implementing the case solution as designed often led to more knowledge and data on the problem, leading me to iterate until I was satisfied. However, due to the linear nature of this document, my process is presented in a linear manner.

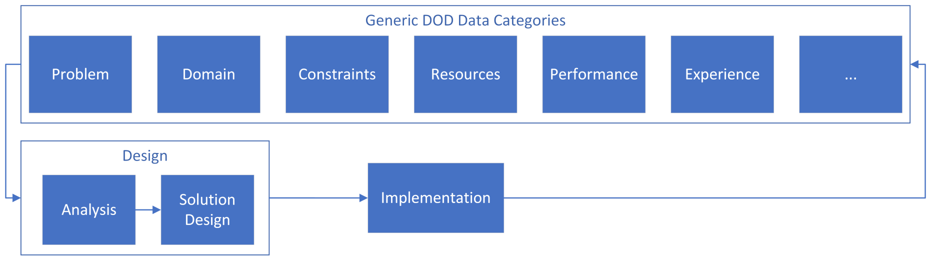


Figure 8: Generic DOD model

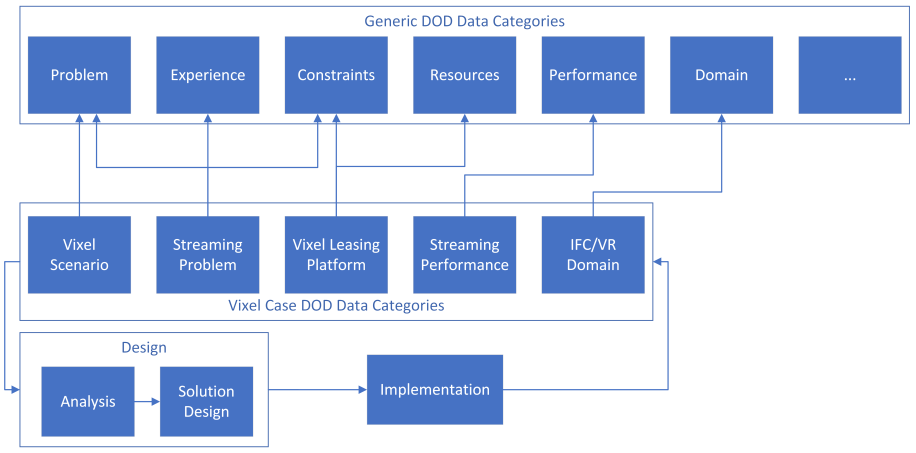


Figure 9: Case specific DOD model with main relations to generic data categories

9.1.1 Guide

Throughout this project I interacted with a practitioner from the game industry. The conversations concerned aspects such as finding resources for benchmarks, dealing with source, runtime, and binary formats, general approaches to streaming within video games, and general approach to solving the case. However, the actual implementation was done independently.

9.2 Data Gathering

Before beginning development I analyzed the case for different pieces of information/data that could be taken advantage of or that needed to be kept in mind. For the purpose of this section I classify them into various subgroups.

9.2.1 Vixel Scenario

Vixel scenario data is knowledge/data found directly in Vixel's case description and requirements, and links to the generic problems category.

No Control Over the IFC Files

Vixel has no control over the IFC files that are sent to them from customers. They can come in all shapes and sizes. This is different from how the situation often is in the games industry. There the creators of the video games control most of the content, and can instruct the artists to create resources that fit within the limits of the technology they are working on.

Existing Build Step

VREX already contains a separate build step, and there does not seem to be a plan to abandon that concept.

Models are Static

The IFC models that users can visit within VREX are static from the point that they are converted.

Read Only Models

VREX does not offer any possibility of modifying the IFC models themselves.

No Need for Low Polygon Models

Vixel did not require any form of mesh simplification for the LOD mechanism. It was enough that the different IFC objects disappeared.

Teleportation Based Movement

Users in VREX can move around utilizing teleportation, meaning that it is hard to predict what location a user is headed towards until they actually get there.

9.2.2 IFC/VR Domain

Domain data is knowledge/data about the domain, in this case, knowledge about the IFC standard, and the fact that we are working with a VR application.

Large Amounts of Graphical Data

The triangulated IFC models are often very detailed, leading to large amounts of graphical data. Earlier experimentation where IFC models were converted to .obj[1] and .mtl[2] formats sometimes lead to 12-14 GB worth of files, and that was with mesh reuse mechanisms in an attempt to reduce the number of duplicate meshes. This is partly due to the level of geometric detail, as seen in Figure 10 and Figure 11. Multiple instances of both the sink and the showerhead in those figures are contained within the building shown in Figure 12.

IFC Guids are 22 ASCII Characters

IFC Guids are defined to be 22 ASCII characters, and uniquely define an object within an IFC file[65].

[1]http://www.martinreddy.net/gfx/3d/OBJ.spec
[2]http://paulbourke.net/dataformats/mtl/

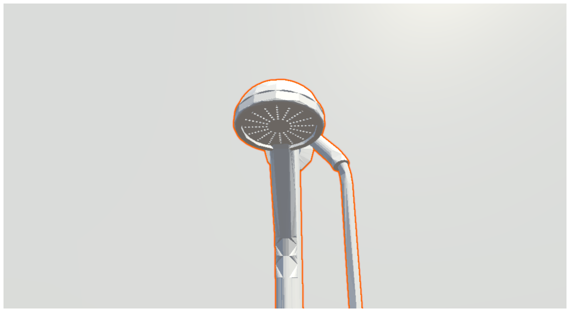


Figure 10: Detailed Showerhead from model[64]

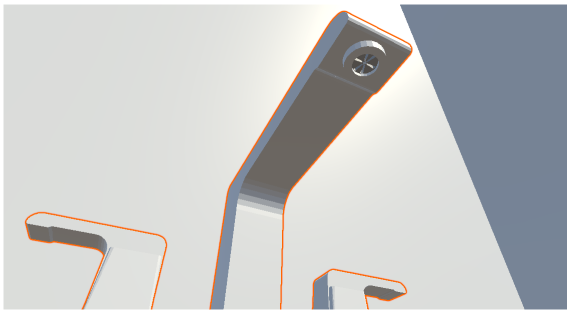


Figure 11: Detailed Sink from model[64]

IFC Objects Vary in Size

The IFC Objects themselves can vary in size, for instance walls within a building can be quite large, however fire alarm and power sockets are usually small objects.

Multiple IFC Files of the Same Model

It is not uncommon for the AEC engineers working on an IFC model to split it into multiple IFC files. For example, one IFC file can contain all pipes and heating facilities in a building, while another file can contain all the walls, roofs, and furniture.

High Framerates Helps Against Cybersickness

The possibility of the user experiencing VR Sickness, also known as cybersickness, is a problem developers face in many VR applications. Many factors influence the possibility of a user experiencing cybersickness. However, the most relevant factor for this case is framerate, as low framerates are known to increase the chance of experiencing cybersickness[66].

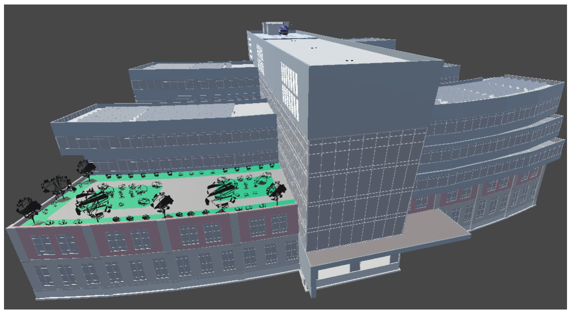


Figure 12: Visualization of architecture and structural layer of the model[64]

9.2.3 Vixel Leasing Platform

Vixel allows customers to lease hardware from them. Vixel leasing platform data is knowledge/data regarding the constraints and resources of the hardware that the application will run on.

Expected Hardware

The computers that customers lease through Vixel is considered "expected hardware", and the streaming solution is supposed to be written with those in mind. Table 11 shows a selection of relevant hardware information.

Processor	
CPU	Intel Core i7 (8th gen) 8750H
Number of Cores	6
Instruction Set	64-bit
RAM	16GB (8GB as minimum spec)
Disk configuration	512 GB SSD - (M.2) PCIe - NVM Express (NVMe)
Operating System	Windows 10 Home (64-bit)

Table 11: Relevant Expected Target Platform Info[67]

VR Headset Update Time

The headsets targeted for the streaming solution have an update frequency of 90 Hz. The update frequency of the VR goggles impacts the frame budget. We get $\frac{1s}{90Hz} \approx 11ms$ time per update.

9.2.4 Streaming Problem

The streaming problem data is data gained from a high level analysis and experience of what problems needs to be solved and what constraints they need to be solved under.

Problem Analysis

Deciding what to stream

The solution needs to figure out which objects should be loaded and unloaded, this should be based on the users current location. The logic for solving this problem will run every frame, and should therefore be relatively well performing and predictable.

Streaming in Graphical Representation

The solution needs to load in the graphical representation of the IFC objects from disk. This situation does not occur as regularly as deciding what to stream. However, for the users experience this should happen fast, and I need to avoid hogging the main thread. The graphical representation of the IFC objects are transparent or opaque meshes. Coloring the meshes are done through vertex colors.

Query IFC Objects

The solution needs to allow the user to select an object in the virtual world by pointing and clicking on it to get more IFC data regarding the selected object. To be able to select an object we need to do a raycast to find out which object the user is aiming at, which requires the use of mesh colliders. The user selecting an object is not part of the main loop, and there are no hard deadlines, but raycasts can be expensive, so it is important to avoid hogging the main thread.

Streaming in Mesh Colliders for Raycasting

As with the graphical representation the mesh colliders should be streamed in, both to avoid unnecessary memory and runtime overhead, and to stop the user from being able to select objects that do not have a graphical representation in the world. Streaming in the mesh colliders does not have as strict deadline as the graphical representation as it is unlikely that the user will select something the moment streaming starts. Similarly, it is not a large problem if the user selects the wrong object, as they can simply re-select the correct one. As with the other problem areas, it is important that streaming in the mesh colliders avoids hogging the main thread.

9.3 Solution Design & Acting on data

9.3.1 Splitting the Model

While extracting the above data I was also communicating with both my DOD guide and Vixel regarding how the models could be split up into streamable units that could be loaded into the application at runtime.

Different questions were discussed regarding what a streamable unit could actually be. What shape would it have? What would it contain? Would I only need one type of streamable unit, or should I have multiple based on the IFC Object types (one for architecture, and another one for interior)? How should I deal with spatially large IFC Objects? How should the streamable unit be guided by the disk bandwidth and size of content?

In the end a multi-layered voxel grid approach was chosen. The graphical representation of the IFC model would first be split into different layers, based on the physical size of the different objects. We decided on having three layers of varying size, and a fourth layer that would contain any object that would not fit within the other three layers. Each layer consisted of a 3D voxel grid, with the cell sizes determined by the layer. Each object were placed within one of these cells based on their position and size. This can be seen in Figure 13, Figure 14, Figure 15, and Figure 16.

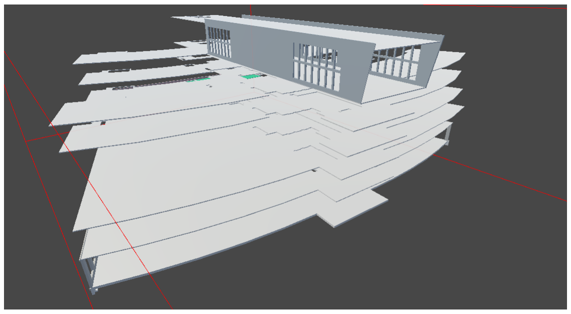


Figure 13: All objects[64] within the top layer (those who cannot fit elsewhere)

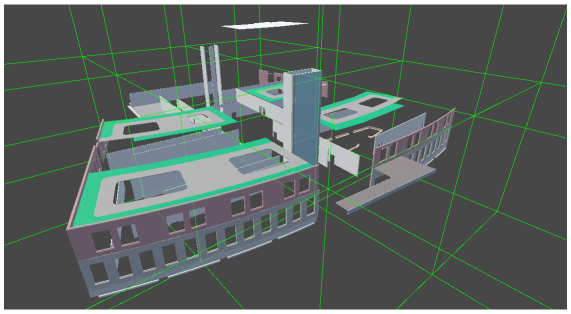


Figure 14: All objects[64] within the first voxel layer (green voxels, 48Mx48Mx48M)

Each voxel is considered a streamable unit, and at runtime, the solution would find the voxel containing the user in the smallest layer. That voxel, the 26 adjacent voxels, and all voxels containing those voxels at the larger layers would be streamed in. The fourth layer would be loaded on startup and stay within the application at all times.

This approach was chosen due to several factors, such as simplicity, and that it trivially handles large IFC objects without requiring them to be split into multiple shapes. Additionally, it gives the user a constant measurement of how far they can see around themselves; 1.5 of the length of a

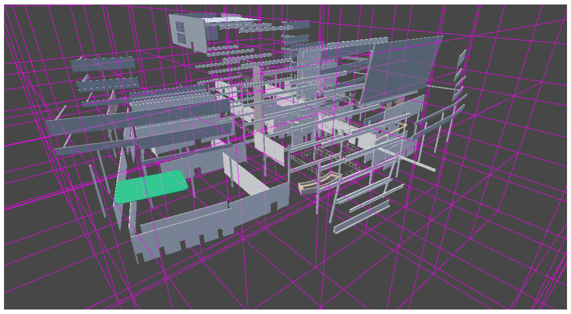


Figure 15: All objects[64] within the second voxel layer (magenta voxels, 24Mx24Mx24M)

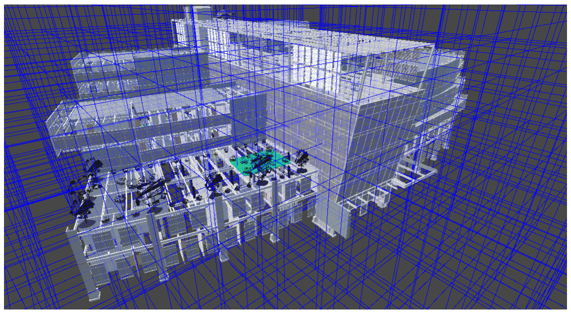


Figure 16: All objects[64] within the third voxel layer (blue voxels, 12Mx12Mx12M)

voxel in the lowest layer. Lastly, this grid can also be used to find the location of the user, and its workload is very predictable.

9.3.2 Voxel Size & Streaming Budget

Following the decision on how to split the model into voxels, we needed to decide on how large the voxels should be. This decision was to be guided by the worst case read bandwidth of the drive and the acceptable duration to stream in the geometry specified by Vixel.

To find the bandwidth I ran a benchmark on the target hardware[3]. Both sequential and random read speeds were investigated and the lowest number gave me the worst case. 30MB/s[4] was chosen as the worst case, based on the benchmark and some room for performance loss.

After deciding on the worst case drive bandwidth I consulted Vixel regarding how long it should take to load in the needed geometry. We decided that going from no geometry to all the geometry contained within the voxel the user is standing in, the 26 adjacent voxels, and all voxels containing them at higher levels, should not take more than 0.5 seconds. This time frame means that in no

[3]Results: `https://www.userbenchmark.com/UserRun/17271175?redirFrom=userbenchmark.com&`
[4]30MB/s was too low and lead to pessimistic budgets, this is discussed in reflections

situation can we stream more than $30MB * 0.5s = 15MB$[5] from disk.

Since we have no control over the IFC files we receive from users, 15MB[6] cannot be used to determine the voxel sizes. Our idea was that the converter could use that number to decide on a fitting voxel size. However, we decided on a minimum voxel size of 3Mx3Mx3M, as voxel sizes smaller than that would probably not be useful for the end user. Having the converter figuring out the best voxel size was dropped due to the scope of the project. Instead I used the minimum voxel size at the beginning of the project and started systematically experimenting with alternative values when that proved too small to be useful in VR and also felt disorienting. In the end I commonly tested with 12Mx12Mx12M as a minimum value.

9.3.3 Final Design & Implementation

With all the previous data in place, it is possible to discuss the final design and implementation of the solution. Each of the following subsections contains a table with the data I discussed in the previous sections and the action I took based on that data. Selected (*italicized*) data and actions are given a brief discussion. The following sections only discuss the core of the solution, i.e. the logic for loading and instantiating objects from disk. Other parts of the solution such as how voxels are removed or the functionality of the IFCConverter, are not discussed. This is either because the solution was trivial or because there was not a clear "Data $\rightarrow$ Decision" logic as the solution was a mere reaction to previous decisions. The IFCConverter is not discussed as I was mainly doing exploratory programming to understand how to solve the problem of splitting the model, and also calculate various needed stats. Furthermore, it was implemented with standard Unity MonoBehaviors rather than Unity's new data-oriented tech stack (DOTS)[7] and its main functionality data decision points are already covered by the discussion of the file format.

Unity DOTS & Voxel Entity Lifecycle

Since the motivation behind this case is to investigate DOD, it makes sense to try to make use of Unity's new data-oriented tech stack, in particular the Entity Component System (ECS)[8] and Job System[9].

Figure 17 shows the lifecycle of a voxel that is streamed in and out of the application. As seen from the figure, the VoxelStreamRequesterSystem looks for voxels that contains or surrounds the user, and create requests to the MeshStreamerSystem and ColliderStreamerSystem that those voxels should be streamed from disk and into the application. The VoxelStreamRequesterSystem also

[5]Correction: Originally said $30MB * 0.5s = 12.5MB$, error was due to previous estimates using 25MB/s as disk read speed. Unfortunately the error of 12.5MB rather than 15MB continued throughout the document

[6]Correction: Originally 12.5MB

[7]https://unity.com/dots

[8]Explanations and documentation for Unity's ECS can be found at: https://docs.unity3d.com/Packages/com.unity.entities@0.1/manual/index.html

[9]Explanations and documentation for Unity's Job System can be found at: https://docs.unity3d.com/Manual/JobSystemOverview.html

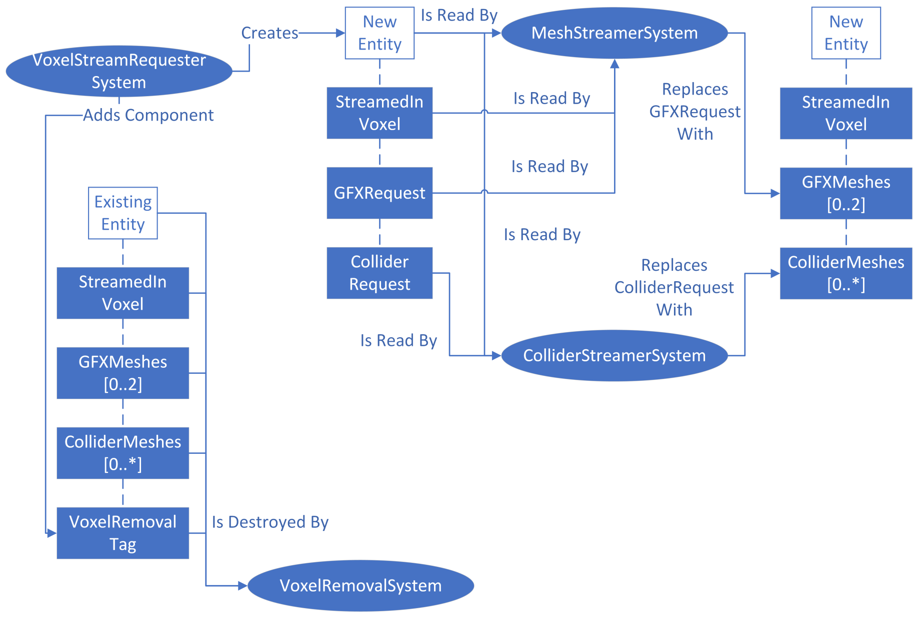


Figure 17: Solution Systems Overview

detects already existing voxels that should be unloaded from memory due to being out of range of the user. These voxels are marked with a VoxelRemovalTag, and removed by the VoxelRemovalSystem.

.str File Format

The .str file format is used to store the data that will be streamed into the scene. It is a binary format, designed to be interacted with through a memory map. An overview of the format can be seen in Figure 18. The decisions used to guide the file format can be seen in Table 12.

The format consists of a, currently unused, header section, a 32-bit integer stating how many layers are used within the model. This is followed by the sections segment, that indicates the offset of the different sections. This in turn is followed by different data and info sections. The Bounds Center Section contains the center position of the various bounding boxes. The combination of this along with the VoxelSizes and Dimensions is used to create the voxel grid used for detecting the users position and thus determining the content to be loaded.

The Bounds Center section, as well as the other data sections are stored in a one dimensional contiguous sequence, requiring a layer, and depth, row, and columns indices for access, as seen in Listing 9.1.

Data as Basis for Decision	Resulting Decision
Existing Build Step Known Target Hardware	Binary File Format
Existing Build Step *Semi-sequential Access* *Known Target Hardware*	*Memory Map*
Data needed in different contexts *Unity mesh accepts vertex and index data in separate containers*	*Sectioned Data/Relational format*
Voxel bounds accessed frequently All bounds in a layer has uniform size	Store just center of each bounds, create bounding boxes and keep them in memory
Varying number of objects in each voxel *Varying number of vertices/indices per object* *Unity colliders take up varying number of bytes*	*Info Structures*
Names are always 22 ASCII characters	No need for Name Info Structure

Table 12: Overview of decisions made on fileformat based on data

```
int CoordsToIdx(int3 coords, int3 dimensions)
{
    return dimensions.y * dimensions.x * coords.z
         + dimensions.x * coords.y
         + coords.x;
}

int offset = 0;
for (int i = 0; i < layer; i++)
    offset += dimensions[i].x * dimensions[i].y * dimensions[i].z;

int idx = CoordsToIdx(coords, dimensions[layer]);
float3 center = bounds[offset + idx];
```

Listing 9.1: Index Lookup

Memory Map

Since we already have a build step, and we know the target hardware, we can make the .str format a binary file format. My experience tells me that a binary file-format would be faster than one that requires parsing text, as it is possible to do simple memory copies back and forth[10]. Since we will be streaming content from disc based on where the user is located and the user can teleport around, it is safe to say that we will not access the file in a predictable manner. However, once we do find the data we are interested in, we will be reading mostly sequentially. To simplify access, and to avoid overhead of system calls, the file format was created with the intention that it would be accessed through memory mapped IO.

[10]This experience data is the reason for the connection from "Streaming Problem" to "Experience" in Figure 9

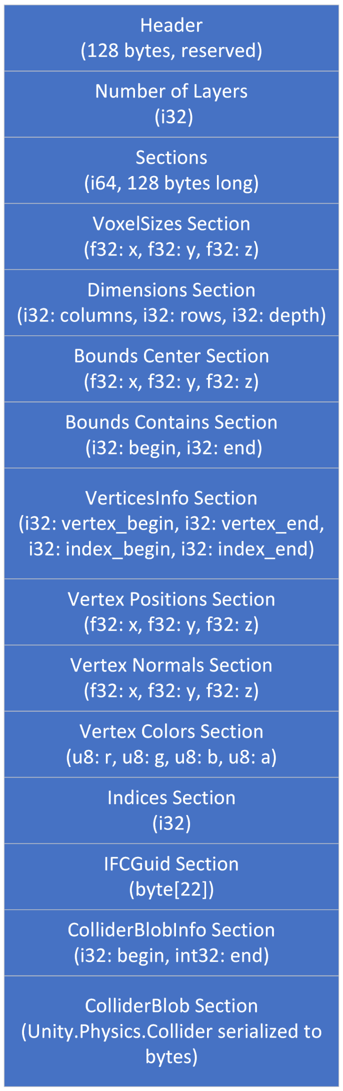


Figure 18: .str file format layout

Relational Structure & Info Objects

The file format follows a relational structure, meaning that the different objects have their various attributes stored in different locations, tied together by sharing an index, this can be seen in Figure 19. Spreading the data like this was decided based on how the different data is used in different contexts. During application startup the application will load in all bounding boxes to create a search able voxel grid. In this situation, the number of objects contained within a voxel and their respective offsets are not relevant, and reading them in at this point would lead to wasted bandwidth. Similarly, once we have figured out where the user is, and want to find out the contextually relevant objects, we do not want to waste our read operations by loading in the center position of a voxel. Unity's Mesh[68] class also expects vertices, indices, colors and normals in separate containers, which is why those variables are not interleaved.[11]

This approach works well when all objects have the same number of values, but it gets a bit more complicated if that is not the case. The info objects (Bounds Contains, VerticesInfo, ColliderBlobInfo) solves this problem by adding another layer of indirection. Essentially, the info objects holds the location of where to find the data of variable length. In Figure 19 Voxel 0 contains three objects (Object 0, Object 1, Object 2), while Voxel 1 is empty, and Voxel 2 contains Object 3. Since an IFCGuid is always 22 characters long, we do not need any info object to indicate where that value is located and how long it is. However, mesh data (vertices and indices) and the size of a serialized ColliderBlob can vary, which is why the VerticesInfo and ColliderBlobInfo attributes are used.

A nice feature of this approach to structuring the file format is that it is trivial to add new attributes without breaking backwards compatibility. An extension would merely require the addition of a new section and ensuring that the new attributes are sorted in a manner such that they share their index with the object they belong to. In combination with the memory map this sectioning and use of info objects also leads to very simple code as all read access to the different sections in the file is done through pointer referencing, as if the different sections were plain arrays.

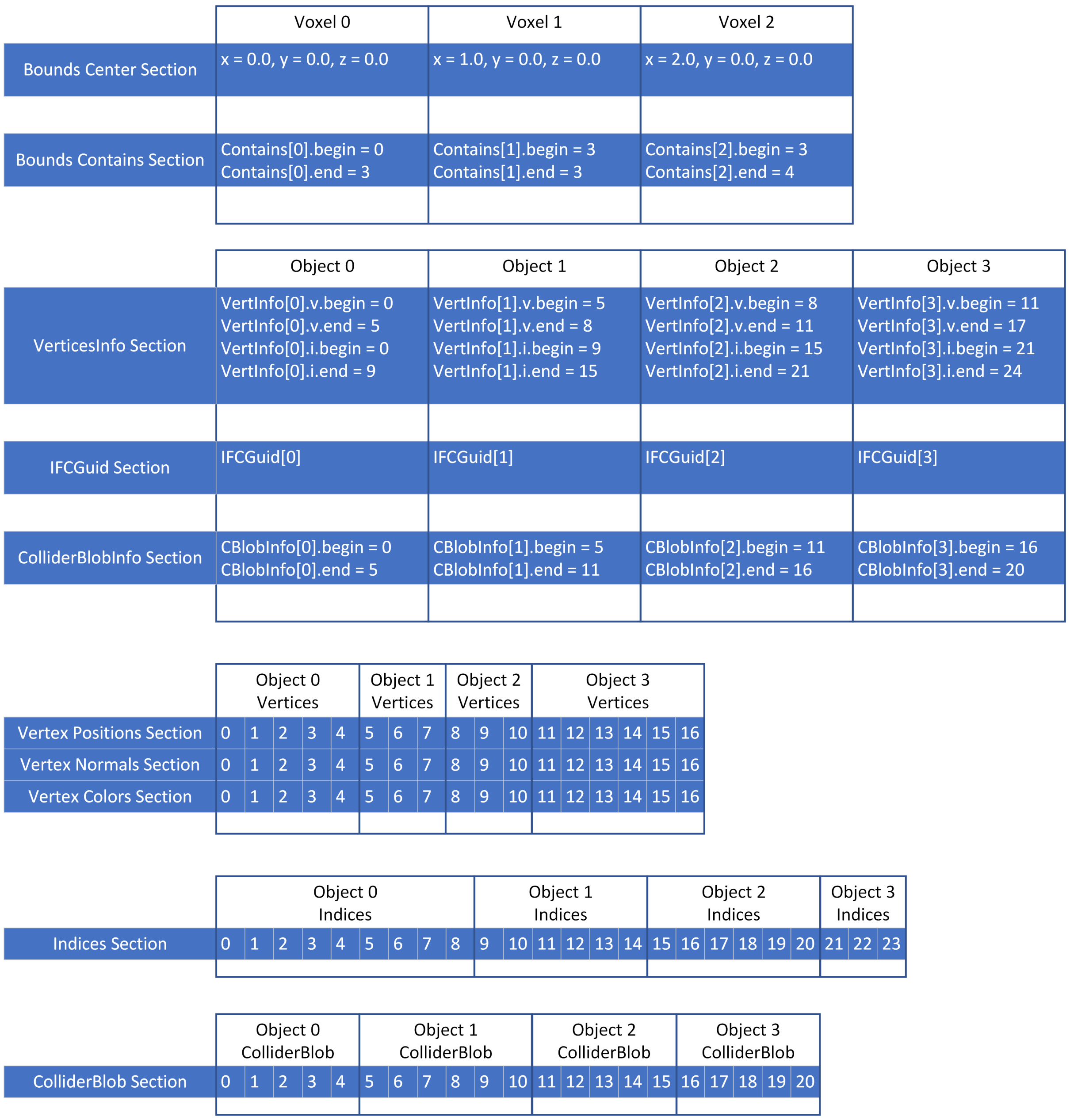
Deciding What to Stream

Data as Basis for Decision	Resulting Decision
Voxel sizes are always a multiple of the voxels in the layer beneath	*Shrink search space after finding outermost voxel when searching for user*
Voxel sizes are always a multiple of the voxels in the layer beneath	*Shrink search space when finding which outer voxels contain inner ones*[12]

Table 13: Decisions made regarding deciding what to stream based on data

[11]This is not the case for how Unity treats the vertex attributes internally, discussed on p. 108

[12]Correction: Originally stated "Shrink search space when adding outer voxels from inner ones", altered for clarity



	Voxel 0	Voxel 1	Voxel 2
Bounds Center Section	x = 0.0, y = 0.0, z = 0.0	x = 1.0, y = 0.0, z = 0.0	x = 2.0, y = 0.0, z = 0.0

Bounds Contains Section	Contains[0].begin = 0
	Contains[0].end = 3
		Contains[1].begin = 3
		Contains[1].end = 3
			Contains[2].begin = 3
			Contains[2].end = 4

	Object 0	Object 1	Object 2	Object 3

VerticesInfo Section	VertInfo[0].v.begin = 0
	VertInfo[0].v.end = 5
	VertInfo[0].i.begin = 0
	VertInfo[0].i.end = 9
		VertInfo[1].v.begin = 5
		VertInfo[1].v.end = 8
		VertInfo[1].i.begin = 9
		VertInfo[1].i.end = 15
			VertInfo[2].v.begin = 8
			VertInfo[2].v.end = 11
			VertInfo[2].i.begin = 15
			VertInfo[2].i.end = 21
				VertInfo[3].v.begin = 11
				VertInfo[3].v.end = 17
				VertInfo[3].i.begin = 21
				VertInfo[3].i.end = 24

IFCGuid Section	IFCGuid[0]	IFCGuid[1]	IFCGuid[2]	IFCGuid[3]

ColliderBlobInfo Section	CBlobInfo[0].begin = 0
	CBlobInfo[0].end = 5
		CBlobInfo[1].begin = 5
		CBlobInfo[1].end = 11
			CBlobInfo[2].begin = 11
			CBlobInfo[2].end = 16
				CBlobInfo[3].begin = 16
				CBlobInfo[3].end = 20

	Object 0 Vertices					Object 1 Vertices			Object 2 Vertices			Object 3 Vertices					
Vertex Positions Section	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Vertex Normals Section	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Vertex Colors Section	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

	Object 0 Indices									Object 1 Indices						Object 2 Indices						Object 3 Indices		
Indices Section	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

	Object 0 ColliderBlob					Object 1 ColliderBlob						Object 2 ColliderBlob					Object 3 ColliderBlob				
ColliderBlob Section	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Figure 19: Relational format illustration

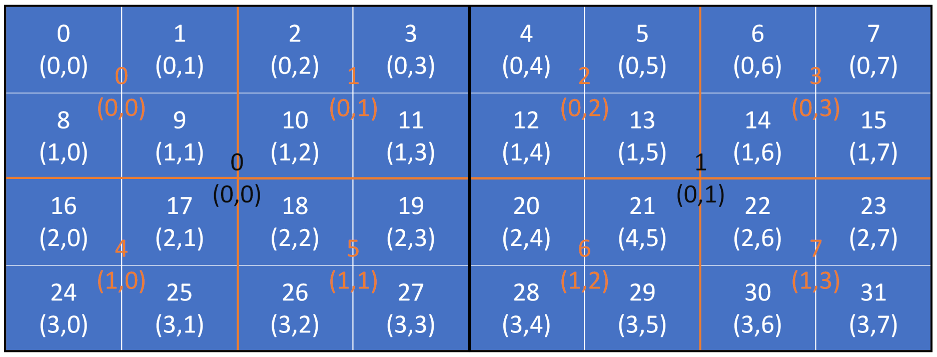


Figure 20: Voxel containment black = layer 1, orange = layer 2, white = layer 3

The logic for deciding what data to stream is done in three steps, and takes advantage of the data seen in Table 13. The steps are executed by the VoxelStreamRequesterSystem. First, find out what voxel the user is currently residing in. Second, find the 26 adjacent voxels, and all voxels containing those voxels. Third, mark voxels that are not already in the scene as voxels that should be streamed in, similarly, mark voxels in the scene that should not be there anymore as voxels that should be unloaded. As previously mentioned the entire model were split into a multi-layered voxel grid. While the voxels in each layer had uniform sizes, the sizes between the layers varied. However, the size of each layer would always be a multiple of the layer above it, which was taken advantage of in several of the algorithms discussed below.

Finding User Voxel

Finding the smallest voxel containing the user is done through multiple linear searches of the various voxel layers. This sounds inefficient, however, by taking advantage of the fact that the number of voxels in a lower layer is always a multiple of the number in a higher layer, we can easily reduce the search space to something much more manageable.

This can be explained easier by an example. Let us assume that the user is located in a position contained by voxel 31 (3,7) in the lowest layer in Figure 20. We would start at the top layer, and find that the user were contained in voxel 1 (0,1). With this knowledge we can calculate the range of voxels we need to search in the next layer. First we calculate the ratio of change between the dimensions of the two layers $(2,4) \div (1,2) = (2,2)$. With this we can calculate the first index in the subgrid of voxel 1 (0,1), $(0,1) * (2,2) = (0,2)$. Calculating where to end is done by adding the ratio to the first index $(0,2) + (2,2) = (2,4)$. Iterating from Voxel 2 (0,2), voxel 3 (0,3), voxel 6 (1,2), and voxel 7 (1,3) will result in voxel 7 being the one that contains the user. Utilizing the previous logic again, we will then search through voxel 22 (2,6), voxel 23 (2,7), voxel 30 (3,6), and voxel 31 (3,7), which is where we will find the user.

This strategy is exactly the same as that employed in the solution, with the exception that it is done in a three-dimensional grid, rather than a two-dimensional one. The worst case situation that can arise is that the user cannot be found within any of the voxels in the outermost layer, as the complexity then becomes $\mathcal{O}(n)$. However, from observation, the number of voxels in the outermost layer are well within ranges that are reasonable to iterate through.

Finding Containing Voxels

Once we have figured out what voxel contains the user, we need to add the 26 adjacent voxels on the lowest layer, as well as all the voxels containing them on the higher layers. Calculating which voxels in the higher layers contain any one of the 26 adjacent voxels follows a similar logic to the previous section, but in reverse.

If we use the same example as before with voxel 31 (3,7) being where the user is located, we need to find out which voxels contain the voxels 22 (2,6), 23 (2,7), 30 (3,6), and 31 (3,7). Calculating which voxel in layer 2 holds voxel 31 is done through integer division of the voxel and ratio between the two layers, i.e. $(3,7) \div (2,2) = (1,3)$, meaning that it is voxel 7 (1,3) that contains voxel 31 (3,7).

This approach has the problem that there will be duplicate entries for multiple voxels. This was handled by storing the voxels in a hashmap. All of the voxels stored within this hashmap is considered to be "relevant" voxels.

Marking Voxels for Streaming & Removal

After finding all the relevant voxels above we need to figure out which of them should be streamed in from disk, as there might be the case that some of them are already loaded into memory. Similarly, any voxels that are already loaded into memory that are not among the voxels discovered above, need to be removed from memory.

This is done through a simple "set" operation. We iterate through all the existing streamed in voxels within a scene, and check if they already exist within the relevant voxels. Any existing voxels found within the relevant voxels are removed from the relevant voxels collection, as they have already been loaded into memory. Any existing voxels that does not exist within the relevant voxels are marked for removal with a VoxelRemovalTag.

Afterwards all the remaining voxels are marked with a StreamedInVoxel tag, GFXRequest tag, and ColliderRequest tag, to indicate to the other systems that they need to be loaded.

Loading Graphical Data

Data as Basis for Decision	Resulting Decision
Read-only Functionality *Only instantiation of meshes must happen on main thread*	*Multi-threaded reading from file*
List is just an array under the hood *Can calculate vertex and index count*	*Pre-allocate space, get pointer to array, memcpy from file into array*
Graphics and Collisions are separate concepts	Combine meshes to reduce draw calls
Costly to create GameObjects	use GameObject pool

Table 14: Decisions made regarding loading graphics based on data

The MeshStreamerSystem is responsible for loading the graphical representation of the IFCObjects from file and into memory. The data guiding its design decisions can be seen in Table 14. The MeshStreamerSystem does this by gathering all voxels with a GFXRequest component, before starting multiple jobs to read the vertex and index data for the objects contained in the various voxels from file. While reading index and vertex data, the different jobs detect whether or not an object is transparent, and combine all opaque objects and transparent objects within a voxel into two different meshes, to reduce the number of draw calls per frame. The opaque and transparent objects needs to be separated to ensure correct rendering order for the transparent objects. Afterwards the MeshStreamerSystem starts to instantiate the voxels on the main thread, which is done over multiple frames in an attempt to avoid large framerate drops. The gameobjects are instantiated from a pool to avoid wasting time on creating new gameobjects.

Multi-threaded reading from file & List Manipulation

While the instantiation and transfer of data from file to the meshes needs to happen on the main thread, reading from file and combining the meshes can happen in other threads. This is done by utilizing the LoadVertices job. However, moving this functionality into the job system required jumping through multiple hoops.

Unity's Mesh[68] accepts data through managed arrays, alternatively they also accept data through non-allocating C# List functions[69]. The problem with these two ways of accepting data is that Unity's job system requires that a job only contains blittable types[70]. Storing a C# List or a managed C# array in a job would stop it from being blittable. It is possible to get pointers to managed arrays in C#, and pointers are blittable types, but creating managed arrays everywhere would lead to lots of unnecessary allocations when communicating with the mesh interface.

However, a List in C# is a wrapper for an array and a size variable[71], and it is possible to use reflection to get access to that internal array, which we can then get a pointer to. Utilizing this trick to get pointers to a list's internal array allows us to read directly from file into the lists that will be sent to the meshes we want to instantiate.

However, when we only have a pointer to the internal array, we cannot dynamically grow the array in an easy manner. This is fixed by simply pre-allocating enough space in the various lists before getting the pointer to the internal array. How much space we need to allocate can be read from the VerticesInfo section in the file we are streaming from.

Lastly, Unity's job system analyses jobs to detect race conditions, this is not possible when pointers are involved, which is why they are not allowed by default[72]. However, using the NativeDisableUnsafePtrRestriction attribute you are allowed to put pointers into jobs at your own risk. Race conditions are avoided in the LoadVertices job by ensuring that each job writes to their own list.

Loading Collider Data

Data as Basis for Decision	Resulting Decision
Read-only Functionality *Can create colliders on separate threads*	*Multi-threaded reading from file* *& Instantiation*
Colliders will not move after instantiation	Attach frozen attribute
IFCGuid is read-only and rarely needed	*Store as direct pointer into memory map*

Table 15: Decisions made regarding loading colliders based on data

The ColliderStreamerSystem is responsible for loading and instantiating the MeshColliders for the different IFCObjects from file. The decisions guiding its design can be seen in Table 15. Similar to the MeshStreamerSystem the ColliderStreamerSystem gathers all voxels with a ColliderRequest component, and starts multiple jobs for loading all the colliders from file. The MeshColliders, unlike the meshes used by the MeshStreamerSystem, can be instantiated outside of the main thread, allowing for full parallelization. Once all the MeshColliders have been loaded, entities for the IFCObjects are created, and the MeshColliders are attached as components. Additionally, the IFCGuid component is attached to the different entities so it is possible to identify them when doing raycasts. Translation and rotation components are also added to the different entities. Lastly, a Frozen component is added, to indicate that this entity will not change its transform, and should therefore not be updated.

Multi-threaded Reading From File & Instantiation

From inspection of the Unity entities and Unity physics package source code, deserialization of colliders is thread safe, which makes it easier to parallelize. Since this system mainly used Unity DOTS functionality there was no need to jump through as many hoops to get access to the data we needed as it was in the MeshStreamerSystem. This meant that reading the data from the file was jobified by simply giving each job a pointer to the relevant ColliderBlobInfo and ColliderBlob sections of the memory map, and an output array to write the colliders to. Instantiating the entities and other components after the collider data had been read was done through the PostUpdateCommands buffer supplied by Unity DOTS to avoid array invalidation as a result of entity updates.

IFCGuid Component

The IFCGuid component is used to identify what IFCObject a collider represents. This component is only relevant for the user of the application, and only in the situation where they have selected something. Since this component is required so infrequently and accessing it does not have a hard deadline, it is simply stored as a pointer into the memory map, rather than as a string. There is no need to duplicate all the IFCGuids as strings when fast access is not required.

9.4 Result

In the end, the development was a success. A video showing the VR functionality can be seen at https://www.youtube.com/watch?v=jJToc9uKA00, while https://www.youtube.com/watch?v=gEE3nMjxk_Y shows how different parts of the model are streamed in and out of the application. The videos are made for showing how various parts of the world are streamed in and out. In real use of a BIM inspection tool such as this, the user would probably stand more still at the various location to perform their inspections. The video shows the West Riverside Hospital model[64], which has been split into voxels of voxels of sizes 48Mx48Mx48M, 24Mx24Mx24M, and 12Mx12Mx12M. The solution streams in three voxels in each direction rather than just one, i.e. all content within $12 * 3 + 12/2 = 42$M in each direction from the center of the voxel the user is standing in.

A repository with the source code for the project can be found at https://github.com/Per-Morten/master_project.

9.4.1 Meeting Requirements

As demonstrated by the video, multiple of the requirements have been met. An overview of how these requirements have been fulfilled can be seen in Table 16. Each requirement is discussed in the following paragraphs[13]. The solution dynamically loads and unloads objects based on position, as required by requirement 1, which then also means it supports the LOD mechanism of requirement 4.

Requirement 2 is marked as fulfilled as it will still be possible to hide different objects. However, due to the combination of meshes, it will require explicit modification to the index buffers of the various meshes.

The solution only requires the position of the user, meaning that while no other movement options but teleportation have been tested, it should have no problem with supporting other movement options, which is why requirement 3 is marked as fulfilled.

The solution also allows a user to query for IFC objects as per requirement 5.

Requirement 6 was vague originally. In the cases where we are not streaming in any data, the

[13]Correction: Change from "Whether or not these requirements have been fulfilled can be seen in Table 16", and added a sentence for clarity

VoxelStreamRequesterSystem operates at around 1-4 ms per frame depending on how many voxels we are streaming in in each direction. This was deemed as within acceptable ranges by Vixel, which is why it is marked as fulfilled. Additionally, we are well within the pessimistic stream budget of loading in 15MB[14] in 0.5 seconds.

Lastly, requirement 7 is marked as fulfilled as this document describes the core functionality.

Requirement	Fulfillment	Method
1: *The Solution Shall Dynamically Load and Unload Objects Based on a Users Position*	Fulfilled	Demonstration
2: *The Solution Shall Support Existing Hiding and Selection Schemes*	Fulfilled	Analysis
3: *The Solution Shall Support Existing Movement Schemes*	Fulfilled	Demonstration/Analysis
4: *The Solution Shall Incorporate a Level Of Detail (LOD) Mechanism*	Fulfilled	Demonstration
5: *The Solution Shall Allow For Querying IFC Objects for Their Unique ID to Allow for Extra Information*	Fulfilled	Demonstration
6: *The Solution Shall Incur Minimal Overhead*	Fulfilled	Demonstration/Analysis
7: *The Solution Shall be Documented*	Fulfilled	This Document

Table 16: Requirement Fulfillment

9.5 Reflection

Creating a solution for this case has been an educational experience, which is reflected upon in the following sections. First I reflect on the technical solution and opportunities for future iterations, before reflecting over the development experiences.

9.5.1 Reflection on Solution

I went through multiple iterations when developing the solution, before reaching a point I was relatively satisfied with. Through the iterations I have learned a lot, and see lots of openings for further development. However, due to time constraints, I am not able to actually implement all of those ideas. Below is a collection of changes I would like to experiment with if I were to do another iteration on the solution. Their omission from the original solution is mainly due to inexperience and time constraints.

[14] Correction: Originally 12.5MB

Non Unity IFC Converter

The IFC converter was written in Unity to have a visual way of debugging the converting process. However, that comes with a lot of overhead as time is spent not just triangulating, but also drawing the model. Additionally, we do not avoid the issue of models being too large for Unity to handle. It would probably be better to write a converter in C++ or something similar to get faster conversions with even larger models. Due to the use of Unity physics colliders, part of the conversion would still need to happen in the Unity engine, but from experience, creating and serializing the colliders usually took the shortest amount of time, so a lot of speedup could still be gained.

Mesh Interface & Interleaved Vertex Data

I originally operated on the understanding that Unity's Mesh[68] class expects vertices, indices, colors, and normals in separate containers. However, it looks from the documentation[73] that it is possible to get access to the native vertex and index pointers, and that the vertex data is stored in an interleaved format. If I were to iterate further, I would store the vertex data in an interleaved format in the .str file rather than the relational section one. I would also use the native graphics API's to faster copy over memory to the GPU.

Detect Opaque and Transparent Meshes & Combine Meshes in Converter

Currently all meshes within a voxel are combined into two large meshes as they are read from the file. However, this is unnecessary logic, instead all meshes in a voxel should be combined in the converter. Similarly, we detect whether or not a mesh contains transparent vertices while streaming in the mesh data. This should also have been dealt with at the converter stage. The reason combining meshes were not done at converter time was mainly due to inexperience. Detecting transparent vertices at run-time was done due to some issues with the color values I got from the library used for triangulating the IFC files. Running the converter multiple times to test the transparency values were a more time consuming process than loading an already converted file. However, having found alpha values I am happy with I would have gone back and implemented this fix in the converter if I were to iterate once more on this project.

Organizing Voxels Based on Higher Layers for Better Cache Utilization

Currently the voxels in each layer of the voxel grid are stored without any regard for what voxel they belong to in the layer above them. This leads to sub-optimal cache usage, which can be seen from the examples in deciding what to stream[15] and Figure 20. It is highly unlikely that voxel 8 (1,0) is loaded into cache when voxel 0 (0,0) and voxel 1 (0,1) on the same layer has been accessed. Instead voxel 2 (0,2) and voxel 3 (0,3) are probably loaded, which are not needed at this point. If these could somehow be sorted in such a way that voxel 0, 1, 8, 9, would lie contiguous we would have better cache utilization for most of the work done in the algorithms above.

[15] See Section 9.3.3

Move More Functionality out to Jobs

Currently all of the work related to finding out which voxels to stream in or remove happens on the main thread. However, with some adjustments it should be possible to place all this work out on other threads. There just has to be some synchronization ensuring that voxels that have only been partially streamed in are not removed until they are fully loaded. An earlier iteration of this solution experimented with moving the process of finding the containing voxels into multiple jobs. This code was discarded prematurely as it performed worse than the single threaded version, however it would have scaled better when loading more than only 26 adjacent voxels from the user. Another iteration would either have brought this functionality back in, or found another way to ensure that partially streamed in voxels would not be marked for removal, as that would allow this and most other systems to run concurrently.

A possible solution would be to mark voxels that are in the process of being loaded with a GFXInProgress and ColliderInProgress tag, and have entities with those components being excluded from the query run by the VoxelStreamRequesterSystem. This way the MeshStreamerSystem and ColliderStreamerSystem would not have to tell the VoxelStreamRequesterSystem that they are busy working, and that the VoxelStreamRequesterSystem should halt its functionality until they are done.

These solutions probably starts touching on the field of garbage collection, which I do not have experience with. Due to time constraints this link will not be investigated further.

GameObject Renderer

Originally I attempted to make use of Unity's new hybrid renderer, so that all parts of the solution would make use of Unity's ECS. However, early tests showed that the rendering performance of the hybrid renderer was worse than Unity's standard GameObject model, so I chose to go with that instead. Unity's hybrid renderer still requires the use of Unity's mesh class[74], which cannot be worked on within jobs[16]. So even with the hybrid renderer I would not have been able to solve the main issue with this system, which is instantiating the graphical objects. However, the hybrid renderer was only tested when I was drawing each gameobject individually, rather than combining meshes. It would be interesting to see how it would have performed now that I combine meshes to reduce the number of draw calls.

Waiting for all Jobs to Finish

In the systems that utilizes jobs I schedule all the jobs and wait for all of them to finish before moving on to the next phase. For example, in the MeshStreamerSystem I do not start instantiating meshes until all the jobs are finished. A better approach would probably be to instantiate meshes as the various jobs finish, as that would reduce the total time it takes to load and instantiate the

[16]This functionality seems to be quite requested: `https://forum.unity.com/threads/feedback-wanted-mesh-scripting-api-improvements.684670/`

meshes.

Better Load Balancing in Jobs

Related to the point above, the MeshStreamSystem loads each voxel in its own job. This means that voxels containing large amounts of data are only processed by a single thread. It might have been better to detect how much data each voxel contains and try and split the workload more evenly over multiple jobs.

Keep Pointers to Lists Pinned

In the MeshStreamerSystem there is a lot of work pinning and unpinning the internal arrays of the various C# lists. Pinning and unpinning them is needed to avoid them being moved by the garbage collector[75]. Currently I pin them before I start a job, and release them after the job is complete. However, from my understanding, I only need to re-pin any of the internal arrays if they reallocate. Since I am in control over when they reallocate there is no reason to always pin them before a job and release them afterwards.

Instantiating Entities Over Multiple Frames

In the ColliderStreamerSystem the entities holding the colliders are currently instantiated using the PostUpdateCommands buffer, all in one go. There seems to be some overhead related to this, so it might be worth it to look into instantiating the entities of multiple frames instead, similar to how the MeshStreamingSystem works.

Smaller Voxels to Support Culling

The solution struggles with GPU rendering performance in more open areas (as seen in the VR demonstration video). I believe this is due to the voxel sizes being too large. Larger voxel sizes means that smaller objects will be rendered unnecessarily. Additionally, I believe the way I combine meshes makes it much more difficult for Unity to cull the various objects when drawing.

Support IFC Models of Varying Sizes and Offsets in Converter

The IFC converter currently does not support various aspects of the IFC standard. For example it does not respect projects that are not defined in meters. Similarly, IFC projects are usually defined far away from the origin of the coordinate system, which results in floating point inaccuracies (as seen in the VR demonstration video). If I were to iterate once more I would look closer into fixing this issue.

Longer Fadeout Effects on Teleport

As seen in the VR demonstration video there is a short fadeout effect when upon teleportation. This fadeout is not something I implemented, rather it was added by SteamVR[17] when I utilized their teleportation implementation.

Due to the drop in performance when streaming in new objects, we wanted the fade out to last until all objects had been streamed into the scene, stabilizing the framerate. This is functionality I would have added if I were to iterate once more.

Thresholds for Streaming

Streaming in the voxels as soon as a user is within proximity, and streaming it out again as soon as the user has moved out of proximity can cause problematic situations. One such example is if the user is moving back and forth at the border of where a voxel should be streamed in or out. This situation will lead to a lot of unnecessary operations streaming voxels in and out, effecting performance. A better solution would be to have different thresholds for when a voxel should be streamed in, and when it should be streamed out. So that a voxel is not streamed out until the user has gotten further away from it.

9.5.2 Reflection on Methodology & Development

Pessimistic Streaming Budget

As previously mentioned[18] the streaming budget we set for ourselves were to stream in 15MB[19] in 0.5 seconds. The budget was based on benchmarking on target hardware, using the random read speeds; 30 MB/s, as a guiding factor. However, since I make an attempt at relatively predictable access patterns and sequential reads, using the random read speeds as a guiding factor becomes too pessimistic. Instead I should probably have looked at the sequential read speed; 2200 MB/s, and tried to make an adjusted estimated based on that. I detected this mistake late in the project, and therefore decided not to go back and re-adjust the requirement. The main result of this mistake is that the streaming budget does not make sense, and that we are able to deal with voxels that contain more data than we thought originally. An adjusted requirement would not change the way I approached the problem.

Memory Map Issues

Due to inexperience with the C# memory maps, it took a while until I understood how to get a hold of the underlying pointer to the memory map. This made development of the solution relatively difficult as all communication had to happen through the memory map interface, rather than direct pointer manipulation. This also made it very difficult to utilize Unity's job system, as it requires

[17]`https://github.com/ValveSoftware/steamvr_unity_plugin`
[18]See Section 9.3.2
[19]Correction: Originally 12.5MB

attributes to be blittable[70]. These problems went away as soon as I understood how to work with the memory maps as pointers, however, had I known this earlier I would probably have been able to do more in this solution.

Performance Measurements

I used the Unity Profiler[20] and Unity Profile Analyzer[21] to measure performance differences for the various optimizations I set out to do while developing the solution. However, I should have been more systematic in this process, so that I could have shared the performance differences in this document. The reason this was not done was due to various factors, such as not running on target hardware or VR in the beginning of development, and poor source control utilization. Additionally, the benchmarking scene I used for performance measurements in the beginning was built for synchronous instantiation of the streaming data, and therefore were not representative after I moved to asynchronous instantiation. Lastly, I mainly worked with optimizations on the various systems as I were developing them, rather than going back for further optimizations. This was making it hard to create benchmarks that were comparable, as I would always be doing more work in the next iteration of the benchmark.

Unity DOTS, and ECS

This is the first time I have ever worked with Unity DOTS and an ECS architecture. I have made earlier attempts at working in a DOD fashion, but have found it difficult. This was due to inexperience and a lack of resources and guidance on DOD. However, it was also due to working in languages where a DOD approach was not encouraged by the design and functionality of the programming language. I felt that working with Unity DOTS and ECS encouraged me to think more about how I split up my data for the different work I was going to do on it. Working with the ECS also led me to better understand the relational nature of ECS and DOD, as it usually felt almost like I was doing GPU kernel programming on a MySQL query.

Working With a DOD Mindset

Trying to work with a DOD mindset has been an interesting learning experience, leading to various insights.

For example, early in the development I was making scripts gathering various data, such as what the most common object within an IFC model was (walls or wires? fire alarms or columns? etc.). I caught myself thinking in the problem domain rather than the data domain. It does not matter to the disk drive whether it is loading walls, wires or chairs. The only thing that is important is how much space the triangulated data of the various objects will occupy on disk. The data regarding most common objects could have been useful if I were to modify the converter to take advantage

[20]https://docs.unity3d.com/Manual/Profiler.html
[21]https://docs.unity3d.com/Packages/com.unity.performance.profile-analyzer@0.4/manual/profiler-analyzer-window.html

of it, however, that was deemed outside of the scope of the project.

Through this project I have also gained a better understanding of what I can actually expect from my hardware. Initially, I had no clue how fast my computer actually is, or how I could make realistic estimates based on hardware (as indicated by the pessimistic streaming budget). However, I now have mental tools that can help me better understand and estimate performance characteristics of different systems.

9.6 Conclusion

This document has described the implementation of a streaming system for a BIM inspection VR solution. The design and implementation followed a DOD style approach in an attempt to show my understanding of DOD. While not perfect, the solution fulfilled most of its requirements, and lead me to many valuable insights.

10 Validation of Case Process

Following the implementation of the IFC streaming solution, I presented the process document to the industry practitioners I interviewed who indicated that they were interested in case validation, in order to get feedback on my process. This chapter focuses on novel feedback, i.e. not what I have already covered in the reflections section of the process document[1].

10.1 Methodology & Participants

The validation followed the same methodology as the interviews[2], with the participants giving their feedback through an online document. I reached out to the five practitioners who had indicated that they were interested in case validation, out of those, four choose to answer the questions.

Raw Feedback

As with the interviews I asked for permission to publish their feedback, and have included the responses I was allowed publish in the appendix (Appendix J, Appendix K, Appendix L).

10.2 Questions

The questionnaire consisted of the four open ended questions presented below. The intention of the questions was first of all to validate if I had followed a DOD mindset/principles, whether or not I had analyzed the correct set of data, and if the presented figures were an accurate depiction of DOD.

1. Do you think the proposed solution has been developed following DOD principles? Why or why not? (Please highlight potential shortcomings if existing)
2. If you believe the proposed solution does not comply with DOD principles, would the described reflections (Section 2.5.1, p. 27-31)[3] alleviate your concern? If not, please highlight remaining concerns.
3. Is there any data or decisions that could have been considered in the process. If so, which ones?

[1]See Section 9.5
[2]See Chapter 5
[3]Section 9.5 in this thesis

4. Would you consider figure 2.1[4] & figure 2.2[5] an accurate representation of a data-oriented design principles/mindset? Why or why not?

10.3 Responses

10.3.1 Following DOD principles

All of the participants concluded that the solution was developed following DOD principles, however they all highlighted some shortcomings or data points that could have been considered. These are addressed below.

Inexperience, Hard Data Gathering & Input/Output

While Richard Fabian notes that the solution is developed with a DOD mindset, it is also lacking in, what I understand to be, collection and analysis of more hard data. The case focused a lot on design related data, such as the fact that models are static, what context the application is used within, etc. This might be a result of inexperience as Richard Fabian mentions.

> **Richard Fabian –** Yes [the proposed solution has been developed following DOD principles], but it does appear to be developed by someone inexperienced and also inexperienced in DOD, but that's good as it shows some of the pitfalls of the literature around DOD. There's not enough concentration on the basic aspects of data gathering and analysis. I had planned to add a chapter (tentatively named Counting) on how to look at data and analyse what you have and how you can turn it into usable domain knowledge. I feel a second edition will definitely have that chapter now as it could have helped you on your project.

Stoyan Nikolov brought up a point related to hard data gathering, in that there should have been more focus on the IFC files as input data, and the output data that Unity expects. I attempted to gather more analysis of the IFC files at the beginning of the project, but stopped as it led me to think more in the problem domain, rather than the data transformations that were to be executed. Additionally, I was not able to make relevant decisions based on that data. Again this could be tied to inexperience.

> **Stoyan Nikolov –** I feel that the thesis could be more complete if it also covers better the data sets of input and output. That is to give more clearly the IFC input and the data that Unity expects. It would also be interesting to assess what the result would have been if a non-DOD approach was taken and thus measure the impact of DOD was in this particular case.

From my understanding, Balázs Török notes an example of such hard data. In the implementation, meshes were combined to reduce the number of draw calls, but as Balázs Török highlights, no analysis was done on how many draw calls an average model would contain, nor how many the

[4]Figure 8 in this thesis
[5]Figure 9 in this thesis

target platform could deal with.

> **Balázs Török –** There is mention of avoiding too many drawcalls but there is no data about how many drawcalls would an average model contain without the optimization and how many the target platform could deal with efficiently.

Memory Map & File Format

The .str file format and its memory map intent originally seemed like a point of contention for two of the participants. On the one hand Marc Costa considers it to be designed with a clear DOD intent. However, Richard Fabian finds data access to the memory map to be difficult and therefore considers it anti DOD.

> **Marc Costa –** The design of the .str file format shows a clear DOD intent: memory mappable, sections based on the minimum amount of data needed per process, sections that are used together are close or adjacent in the file, etc...

> **Richard Fabian –** I don't understand the lure of the memory mapped solution. Personally, I would find analysing the data access to the mmap to be difficult, which hides some data from me, which feels anti DOD.

I went back to Richard Fabian and asked him to elaborate on his answer, as it was not entirely clear to me why he considered it anti DOD. He pointed out that memory mapped files hides when the disk access is actually happening, due to parts file being paged in and out of memory by the operating system depending on when they were needed. In that way I can see his argument, as what looks to be a simple pointer deference might require disk I/O operations. Marc Costa clarified[6] that just because the file format was memory mappable did not mean that it had to be implemented using memory map functionality. Rather, his points were that the data in the file would not require any post-processing, that the format was optimized for consumption, and that read operations to the file could be optimized if needed.

From their clarifications, I am of the understanding that Marc Costa is discussing the format itself, while Richard Fabian is discussing the chosen implementation of reading the format.

> **Richard Fabian –** When I used mmap, it had no way of finding out or predicting when the file IO was happening nor how much data was transferred. The code I used mapped the memory segment and would load data from disk via the OS by memory faults. These were unpredictable when the data was unaligned to anything specific. I assumed your data was too. Because mmap gave little feedback as to when new memory was accessed, it meant I had to put it on a background thread. Otherwise, it caused main thread spikes when unexpected loads were triggered by memory accesses just beyond what was loaded, and these were unacceptable.
>
> ...
>
> Ultimately, the mmap version puts you in a dead-end of not being able to see the pat-

[6]I originally did not contact Marc Costa for clarification due to time constraints. He clarified himself over email after he was given a copy of the thesis for review. I was allowed to include the clarification in the thesis.

> terns of access, so you get trapped in just writing code which works but can't easily be improved. You can still make guesses, but you will have difficulty producing repeatable results of profiling and analysis. I would still use mmap if I had a lump of data I wanted to load without thinking about it, but it's a bit of a sledgehammer tool and doesn't give you enough information about what it's doing to find out if it is the cause of any issues. I've had to use platform specific profilers to both prove it was a problem and prove it wasn't a problem in different applications and their uses of the technique.
>
> **Marc Costa** – When I mentioned memory mappable file as an application of a DOD mindset, I was thinking about the fact that the data is directly loadable, without the need for post-processing, e.g. pointer fixups. The fact that the file is memory mappable doesn't directly imply that it needs to use the OS functions to load an entire file into memory. You could still issue specific read requests when the memory is needed as an optimization later. The point is that the format is optimized for consumption and that the reads are **optimizable** if needed.

Verification

Richard Fabian's last comment touches on the importance of verifying your assumptions. This was also brought up by Balázs Török who noted the lack of verification on CPU execution times. I did verify with Vixel[7] that the system operated within what we considered a reasonable overhead. However, that could have been clearer in the document. Similarly, there are other aspects that should have been more clearly verified, such as the draw calls and merging optimization mentioned earlier by Balázs Török.

> **Balázs Török** – ... the layout of internal code structures wasn't described. CPU execution of the update code was assumed to be fast enough because it was based on Unity DOTS but this was not verified.

Unity & its Abstractions

Richard Fabian also notes that there is little discussions as to how Unity implements its culling functionality. Which could have given valuable data/insights that would lead to an even better solution.

> **Richard Fabian** – I notice there's no reference to how Unity does its culling, and there's probably quite a few ways to make the rendering run faster if you looked into how it worked. The culling and occlusion systems in an engine often provide even more data you need to consider.

Related to this is Marc Costa's point that Unity's mesh abstracts away many details that could be valuable, such as what format the data is sent to the GPU in.

> **Marc Costa** – Unity's Mesh. When using Unity there are already many layers of abstraction between the user (i.e. the programmer in this case) and the hardware. It is very hard to calculate what the cost of these abstractions are, but it's trivial to see that it is

[7] See Section 9.4.1

> not zero. The Mesh object is an example: how the rendering data is presented to the user might not be the final format that is sent to the GPU, certain optimizations and GPU features might or might not be exposed (e.g. mesh instancing), etc...

Both these comments highlight that I should have done more analysis on the inner workings of Unity, and that I did not go far enough in my input/output analysis. I was working with the mindset that the end of the pipeline was when the mesh data had been transferred to a Unity mesh. Instead, the end of the pipeline should have been when the vertex and index data was transferred to the GPU, if not at the point when a mesh was actually drawn on screen. A lot of time was spent removing the layers of abstraction on the C# lists to really understand how they work and take advantage of that. A similar approach should probably have been taken for Unity's meshes and culling capabilities.

Duplication

Both Balázs Török and Marc Costa notes that duplicated geometry should have been discussed, as that has implications on the design of the streaming system. For example, allowing for features such as instancing. I deemed de-duplication as out of scope, but should still have discussed it in the document.

Positive Aspects

While there were some shortcomings in the application of DOD certain correct applications were also highlighted. Marc Costa notes various positive aspects, Stoyan Nikolov points out that the case lends itself well to a DOD approach, and Balázs Török highlights that while the estimation for the disk read speed was too pessimistic, it was the right approach.

> **Marc Costa –**
>
> - Hardware constraints: the target hardware and problem specific constraints where determined early in the development (HW specs and target disk read speed).
> - The design of the .str file format shows a clear DOD intent: memory mappable, sections based on the minimum amount of data needed per process, sections that are used together are close or adjacent in the file, etc...
> - Pre-allocating the meshes rendering data arrays and blitting the data from the streaming file directly into the destination.
> - Using Unity's DOTS and ECS to implement systems with the minimum amount of data and functionality to perform their job and being easier to perform in parallel.
>
> **Stoyan Nikolov** – I think the solution considers the guiding principles of DOD and applies the correctly. The problem itself is a good example of something that is well suited to DOD-like designs due to the very strict performance requirements and the relative liberty in terms of data structures and APIs that the author had.
>
> **Balázs Török** – The data layout was considered when it comes to the vertex data in the voxels. There was also data that formed the basis for the disk read speed estimation.

> Even if this estimation turned out to be way too pessimistic it showcases a good DOD approach.

Summary

In summary, the development of the streaming solution followed DOD principles. Aspects such as considering and acting on hardware constraints, the context of data usage, and input and output analysis where highlighted as correct applications of DOD. The practitioners were aligned in that they meant *more* data should have been considered, however, they were not always aligned on *what* data should have been considered.

Based on the feedback, there is room for improvement beyond what was mentioned in the reflection of the process document[8]. In many ways I should have gone further in my analysis. More time should have been spent gathering other forms of data, as most of the data gathered could be considered design data. More time should also have been used to break down and understand the tools and abstractions I was working with, i.e. Unity's abstractions and the memory map.

10.3.2 DOD Visualization

The practitioners were more divided on whether or not Figure 8 and Figure 9 were accurate representations of DOD principles/mindset. Balázs Török thought the approach was well represented, but were unsure about the data categories.

> **Balázs Török –** The approach is definitely well represented. I'm not completely sure about the data categories but those are just examples in this case.

Both Stoyan Nikolov and Richard Fabian commented on the fact that the model was very generic, and could fit with any other development methodology. From my understanding, the data categories in particular were too generic, and focused too little on hard data.

> **Richard Fabian –** ... the figures fit fine in any other development model. For them to reflect DOD, I would expect to see some runtime or data analysis picked out specifically. Without some kind of analysis of the real data flowing through the system, they're not very DOD diagrams. The way I perceive the diagrams, I feel like the Data referenced is almost entirely "design data" not actual hard data making up the meshes, or the converted binaries, or the in memory layout of the loaded str files.

> **Stoyan Nikolov –** The picture could be one very high-level view of the DOD mindset but could also apply to any other software design methodology that also takes into accounts all these aspects. I feel that they are missing an important piece – the "data" itself that gives the namesake of the methodology. In the end DOD's goal is primarily performance, so I would suggest building the diagrams with that in mind as a central piece.

Marc Costa noted a lack of data transformation within the figure, which he puts at the center of DOD.

[8] See Section 9.5

> **Marc Costa –** I would not consider those figures an accurate representation of a DOD mindset. The main characteristic of a DOD mindset is that the data transformation is at the center of the process, which doesn't even appear on the figures.
>
> DOD as a methodology is anti-abstraction, it focuses on data transformations by looking at the inputs and outputs and tries to streamline the transformation by exploiting properties of the data. Most of what comprises a DOD mindset is encapsulated in the Analysis and Implementation boxes.

I believe the criticism mainly comes from how generic the visualizations are, and a lack of proper explanation of them within the process document.

In relation to Marc Costa's comment, I consider the data transformation to be included in the problem category of the visualization, as the problem in and of itself is transforming one set of input to an output. The properties of the data to be exploited is the data contained within the problem and other data categories. This should have been made more clear within the process document.

The visualizations are generic as I wished to capture not just the performance view in DOD, but the more extended problem solving view as discussed in Section 5.5. However, based on the feedback from Richard Fabian and Stoyan Nikolov I created a revised version of the figure (Figure 21), which had a more performance focused view on DOD and asked them for their feedback. This revised figure was not presented to Marc Costa or Balázs Török as they either agreed or brought up different concerns regarding the original figures.

Stoyan Nikolov considered the revised figure to be correct overall, but noted that it was not clear what "design data" was, as I had not used that term in earlier discussions with him.

> **Stoyan Nikolov –** I don't understand what "Design data" is. Do you mean the design or functional constraints that the system has or something else?
>
> Overall I find this correct.

Richard Fabian on the other hand, did not see it as any better than the previous models presented. Rather he wanted more explanations as to what the various categories describes, and their relationships.

> **Richard Fabian –** ... I don't think either are clear enough to help someone who doesn't already understand what you are trying to do when designing with data in mind. Each of the boxes would need a little more information as some look like they overlap. For example, is runtime data different from performance data, if it is, then how does it differ from patterns? How is design data different from dataflow(IO)? I might add "limits" to "Hardware". Maybe the diagram simplifies the content too much. Consider writing a sentence or two on each of the aspects or categories to give the diagram more meaning.

My conclusion from the participants feedback is that there might be merit to the model. I followed the model during the case implementation, to which the participants responded that I followed a DOD mindset. I see that the model presented in the process document might have benefited from

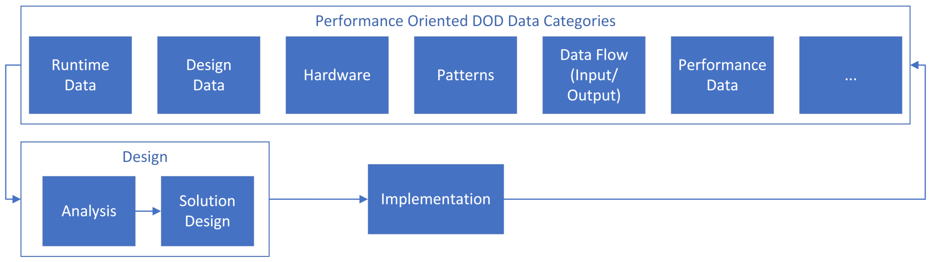


Figure 21: A Performance Oriented Visualization of DOD

a more thorough description, for instance the one in Chapter 6. However, based on the overall feedback, more research is required to create a visualization with unified support, if possible. A possible idea, beyond the scope of this thesis, would be to do more case studies which includes DOD practitioners from the industry, to further develop and validate the model.

10.4 Lessons Learned Regarding Application of DOD in Software Development

Based on my experience when applying DOD to a practical case, and the feedback from the practitioners regarding the process document, I have learned the following regarding the application of DOD.

Deep Understanding of Tools

All tools (programming languages, development environment, libraries, etc.) provide some form of abstraction. When following a DOD mindset, it is important to properly understand how these abstractions impact the solution. Furthermore it is important to understand how these abstractions can be broken down to their underlying data transformations and how those can be taken advantage of. I would argue that the way I removed the layers of abstraction in the C# List is a good example at trying to gain this deeper understanding. The same approach should have been taken with the tools supplied by Unity.

The implementation details of the tools are very important. As with the C# Lists I focused on removing abstractions on the C# memory map until I was left with a simple pointer. However, as noted by Richard Fabian there is still a lot of data hiding left in this tool, as it is not clear when the disk I/O actually happens. In this situation another approach could have been better.

Verify Assumptions

Related to the understanding of the tools is the importance of verifying your assumptions. This can be assumptions about how a tool is implemented, or assumptions about how the hardware works. It is important that you work based on facts, rather than beliefs.

Deep Understanding of Context

It is important to have a deep understanding of the context you are working within. What sort of problem you are trying to solve will affect what sort of data that you should collect, and how that data should be interpreted to reach an optimal solution to your given problem.

Do Extensive Data Collection

Even though I thought I collected lots of different data for the case implementation, all the practitioners highlighted different pieces of data that could or should have been taken into consideration. Such as how Unity implements its culling, statistics on frequency of data contained within meshes, etc. It might be an idea to have some form of tool or framework that gives opportunities for such extensive data collection.

As an answer to research question 3 ("How does one approach software development with a DOD mindset?"), Chapter 6 discussed an initial, theoretical, and quite high level understanding of the application of DOD. I would argue based on my experience when applying DOD to a practical case, and the feedback from the practitioners regarding the process document, that the initial understanding of DOD is on the correct path. However, the description should have had a larger focus on details and verification. Based on the lessons learned, I should also been much more thorough in the application of my understanding in the implementation of the case. This might be related to what Dale Kim and Marc Costa brought up[9], that they had known about DOD for some time until they were able to successfully execute on it.

10.5 Lessons Learned Discussing DOD

Another lesson learned is that it might be best to discuss DOD within the context of cases, rather than at a general level. I base this on how all the practitioners seemed to align in their understanding of DOD in the interviews. However, the practitioners were less aligned in their feedback to the case, highlighting different pieces of data they meant should be considered, and were split in how representative they thought the figure were.

[9]Section 5.2.1

11 Summary & Conclusions

This thesis has investigated the topic of data-oriented design from several different angles, by asking three research questions.

1. To what extent does the conception and discussion of DOD vary between industry and academia?
2. What are the core characteristics of DOD?
3. How does one approach software development with a DOD mindset?

11.1 Research Question 1

Research question 1 was investigated through a literature review of both academic and industry literature (Chapter 4). By analyzing how often various topics were either directly mentioned or implied in the literature, I showed that there is a divide in the conceptualization and discussion of DOD between the industry and academia. The academic literature seemed to view DOD more as a set of patterns, while the industry seemed to view it more as a way to solve given problems.

During the analysis I also identified a set of overarching categories that integrate the different perspectives on DOD in the literature, which can also offer a basis for categorising future work in this area.

While there were some weaknesses in my approach, particularly in relation to the concept of implied mentions, the pattern of the industry touching on a wider range of topics was still present when focusing only on direct mentions. I believe the narrow view held in the academic industry is partly due to the narrow scope and decontextualization that is promoted in the academic profession. Additionally, the types of media used by the industry (conference presentations, blog posts, etc.) lends themselves better to richer elaborations, compared to the academic media (conferences, articles, etc.), which usually center on a particular issue.

11.2 Research Question 2

Research question 2 was investigated both through the literature review and through interviews with industry practitioners (Chapter 5). The interviews followed a semi structured form and focused on various aspects of DOD, such as their understanding of DOD, where to apply it, and benefits & drawbacks. Due to the qualitative nature of the interviews, there were only a few participants, which could be seen as a weakness. Based on the literature review and the industry practitioner interviews, I identified the following aspects as the core characteristics of DOD.

1. Focus on solving your problem at hand, rather than a generic one
2. All kinds of data should be considered
3. Decisions should be made based on data
4. Focus on performance in a wide sense

Another insight gained from the interviews were that many of the participants considered letting go of their OOP mindset to be an obstacle in order to learn DOD. From this I concluded that we should be careful to promote any approach to software development as the "one true way", as that can limit creative problem solving, and lead to fitting a problem to a solution.

The interviews also attempted to gain more insight into the origin of DOD. I were not able to get a proper direct answer regarding the roots of DOD. However, my current understanding is that DOD might have roots in procedural programming and super computers, but that the name and incarnation we see now is rooted in the video game industry. The origin of DOD could be an interesting topic for future research endeavours.

Other topics for future research endeavours from the interviews could also be further research into the experiences brought up by some of the industry practitioners, such as the points regarding simplifying maintenance and testing, reduction in bugs, and impact on productivity.

11.3 Research Question 3

Chapter 6 presented a high-level understanding of DOD, which was meant as an answer to research question 3. I applied this understanding in practice to an industrial case. The case focused on streaming BIM models from disk into a VR application. I wrote a document (Chapter 7 to Chapter 9) explaining my process. The document was sent to willing practitioners in order to get feedback on my understanding and application of DOD (Chapter 10).

The practitioners agreed that I had followed a DOD approach, but pointed out various flaws/limitations in my execution. Such as not spending enough time on data gathering, or breaking down all the abstractions and tools I was working with. I believe much of this was a result of my inexperience with applying DOD.

I also asked the practitioners for feedback on my visualization of DOD, to which I received varying answers. Some where happy with it, others wanted various tweaks. My main conclusion was that there might be merit to the model, but that more research is required to create a visualization with unified support.

In the end I concluded that the high-level understanding of DOD from Chapter 6 is on the correct path, but that the original description should have had a larger focus on details and verification.

Based on the interviews, and feedback to the case I also concluded that it might be best to discuss

DOD in the contexts of cases, rather than a general context.

11.4 Insights

This thesis has shown that there is a divide between the industry and academic understanding of DOD. To me, this suggests that the academic community needs to do more research into the topic of DOD, with a larger focus on collaboration with the industry. The academic literature seems to focused on asking the question of "how" to apply patterns often seen in the application of DOD, when they in my mind should be more focused on the "why".

A tighter collaboration with academia could also benefit the industry. For example, the academic community could help with creating a more unified understanding of what DOD actually is, as the concept still seems somewhat vague, at least from a newcomers perspective. Another benefit would be if students were taught DOD as part of the curriculum, which would make their education more relevant for areas of the industry where DOD is practised.

Even if DOD does not become part of the curriculum, I believe some of the important building blocks of DOD should. These building blocks would in my opinion be:

- Focus on deep understanding of the problem
- Focus on understanding modern hardware
- Approaches to data analysis, applied statistics, and "back-of-the-envelope" calculations
- Deep understanding of how tools can be utilized, but also what, and how, the tools solve various problems.

Lastly, an insight gained from the work in this thesis, is that DOD is a "simple concept that is hard to master".

11.5 Limitations

The most pressing limitations of this thesis is largely the interpretive, and therefore subjective, nature of the chosen methods, both in relation to the literature review, interviews, and case feedback. A possible mitigation strategy would be to conduct inter rater studies in which the interpretation of the interviews is performed by multiple researchers independently so as to maintain a more objective perspective. The interviews and case feedback were also affected by a small sample size. This was due to a lack of access to industrial practitioners of DOD, and the focus on expertise as gauged by public visibility of the practitioners. However, the dedication from the participants should be commended, and shows the commitment to the development of the concept.

A weakness with the case validation and process document is also that it only discussed data that was actually taken advantage of. For example, I did not discuss data regarding factors such as how often various IFC objects was present within the different sample files. This was both because I did not act on that information and due to scope constraints.

Lastly, it is highly unlikely that I got critical opinions on DOD, as I largely interviewed practitioners who in some shape or form were advocating the approach.

11.6 Future Work

Future work could be taken in multiple directions, such as methodological refinements of the study of DOD, or extended investigations of substantive aspects as discussed in the following.

Possible future work could again investigate the existing literature, similar to how I have done, in an attempt to deal with the problematic aspects of the interpretive approach. Similarly, more interviews could be conducted with more participants, in order to deal with the small sample size.

While I have belief in the core characteristics of DOD suggested in Section 5.5, they are largely based on the understanding gained throughout this thesis. A deeper investigation with more rigorous methods would be beneficial for the field.

As previously noted[1] the interview questions were also meant to highlight topics for potential future work. DOD is usually covered from a performance perspective, but it would be useful to look at it from other viewpoints. For instance, does adopting a DOD approach impact the security or testability of the written code? What tools are important and how could existing tools be adapted to fit with a DOD workflow? What are common approaches and best practices for data collection and data analysis when following a DOD mindset?

It could also be beneficial to look at DOD from a more critical standpoint than what has been done in this thesis.

More case studies in cooperation with the industry in order to create a more unified understanding of what DOD actually is is also desirable.

Further investigations into the origin of DOD, as well as its relationship with functional programming could lead to novel insights.

11.7 Concluding Remarks

Even though this thesis has its flaws and limitations, I do believe it achieved its original goal of shining a light on the under-researched topic of DOD, and hopefully establishing a baseline, both in terms of literature analysis as well as providing instructive insights on the development of case studies that lend themselves well for DOD, from which more research can be conducted.

[1] See Section 5.1.3

Bibliography

[1] Unity Technologies. Performance by default. (Retrieved 06.05.2019). URL: https://unity.com/dots.

[2] Ante, J. & Kümmel, M. 2018. Entity component system (ecs) "megacity" walkthrough. Presented at Unite LA 2018. (Retrieved 06.05.2019). URL: https://www.youtube.com/watch?v=j4rWfPyf-hk.

[3] Albrecht, T. 2009. Pitfalls of object oriented programming. Presented at Game Connect: Asia Pacific 2009. (slides only), (Retrieved 08.03.2019). URL: https://web.archive.org/web/20150212013524/http://research.scee.net/files/presentations/gcapaustralia09/Pitfalls_of_Object_Oriented_Programming_GCAP_09.pdf.

[4] Nikolov, S. 2018. Oop is dead, long live data-oriented design. Presented at CppCon. (Retrieved 02.02.2019). URL: https://www.youtube.com/watch?v=yy8jQgmhbAU.

[5] Von Neumann, J. July 1981. The principles of large-scale computing machines. *Annals of the History of Computing*, 3(3), 263–273. doi:10.1109/MAHC.1981.10025.

[6] Fink, A. 2019. *Conducting research literature reviews: From the internet to paper*. Sage publications.

[7] Okoli, C. & Schabram, K. 2010. A guide to conducting a systematic literature review of information systems research.

[8] Given, L. M. 2008. The sage encyclopedia of qualitative research methods. URL: https://methods.sagepub.com/reference/sage-encyc-qualitative-research-methods, doi:10.4135/9781412963909.

[9] Håkansson, A. 2013. Portal of research methods and methodologies for research projects and degree projects. In *The 2013 World Congress in Computer Science, Computer Engineering, and Applied Computing WORLDCOMP 2013; Las Vegas, Nevada, USA, 22-25 July*, 67–73. CSREA Press USA.

[10] Martin, A. November 2007. Entity systems are the future of mmog development – part 2. t-machine.org. (Retrieved 14.06.2019). URL: http://t-machine.org/index.php/2007/11/11/entity-systems-are-the-future-of-mmog-development-part-2/.

[11] Fabian, R. September 2018. *Data-oriented design: software engineering for limited resources and short schedules*. Richard Fabian, 1st edition, Reduced version available from: http://www.dataorienteddesign.com/dodbook/node1.html (Retrieved 03.02.2019).

[12] Martin, A. December 2007. Entity systems are the future of mmog development – part 3. t-machine.org. (Retrieved 14.06.2019). URL: http://t-machine.org/index.php/2007/12/22/entity-systems-are-the-future-of-mmog-development-part-3/.

[13] Martin, A. September 2007. Entity systems are the future of mmog development – part 1. t-machine.org. (Retrieved 14.06.2019). URL: http://t-machine.org/index.php/2007/09/03/entity-systems-are-the-future-of-mmog-development-part-1/.

[14] Homann, H. & Laenen, F. 2018. Soax: A generic c++ structure of arrays for handling particles in hpc codes. *Computer Physics Communications*, 224, 325 – 332. URL: http://www.sciencedirect.com/science/article/pii/S0010465517303983, doi:https://doi.org/10.1016/j.cpc.2017.11.015.

[15] Abel, J., Balasubramanian, K., Bargeron, M., Craver, T., & Phlipot, M. 1999. Applications tuning for streaming simd extensions. *Intel Technology Journal Q*, 2, 1999.

[16] Fabian, R. September 2013. *Data-Oriented Design*. Richard Fabian, online edition, (Retrieved 01.02.2019). URL: http://dataorienteddesign.com/dodmain.pdf.

[17] Llopis, N. September 2010. Data-oriented design now and in the future. *Game Developer Magazine*, 17(8). Also available from: http://gamesfromwithin.com/data-oriented-design-now-and-in-the-future. (Retrieved 01.02.2019).

[18] Llopis, N. September 2009. Data-oriented design (or why you might be shooting yourself in the foot with oop). *Game Developer Magazine*, 16(8). Also available from: http://gamesfromwithin.com/data-oriented-design. (Retrieved 01.02.2019).

[19] Acton, M. 2014. Data-oriented design and c++. Presented at CppCon. (Retrieved 02.02.2019). URL: https://www.youtube.com/watch?v=rX0ItVEVjHc.

[20] S. Liechty, D. 06 2014. Object-oriented/data-oriented design of a direct simulation monte carlo algorithm. volume 52. doi:10.2514/6.2014-2546.

[21] Fontana, T., Netto, R., Livramento, V., Guth, C., Almeida, S., Pilla, L., & Güntzel, J. L. 2017. How game engines can inspire eda tools development: A use case for an open-source physical design library. In *Proceedings of the 2017 ACM on International Symposium on Physical Design*, ISPD '17, 25–31, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/3036669.3038248, doi:10.1145/3036669.3038248.

[22] Sharp, J. A. September 1980. Data oriented program design. *SIGPLAN Not.*, 15(9), 44–57. URL: http://doi.acm.org/10.1145/947706.947713, doi:10.1145/947706.947713.

[23] Netto, R., Fontana, T. A., Fabre, S., Ferrari, B., Livramento, V., Barbato, T., Souto, J., Guth, C., Pilla, L., & Luís, J. November 2018. Ophidian: an open-source library for physical design research and teaching. WOSET. URL: https://woset-workshop.github.io/PDFs/a25.pdf.

[24] Fontana, T. A., Almeida, S., Netto, R., Livramento, V., Guth, C., Pilla, L., & Güntzel, J. L. 2017. Exploiting cache locality to speedup register clustering. In *Proceedings of the 30th Symposium on Integrated Circuits and Systems Design: Chip on the Sands*, SBCCI '17, 191–197, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/3109984.3110005, doi:10.1145/3109984.3110005.

[25] Osborn, J. C. *Operationalizing Operational Logics*. PhD thesis, UC Santa Cruz, 2018.

[26] McKendrick, H. Analysis acceleration in tmva for the atlas experiment at cern using gpu computing. Master's thesis, University of Edinburgh, 2011. Available from: https://www.inf.ed.ac.uk/publications/thesis/online/IM111037.pdf.

[27] Karlsson, M. Evaluation of object-space occlusion culling with occluder fusion. Master's thesis, Mälardalen University, 2011. Available from: http://www.diva-portal.org/smash/get/diva2:443366/FULLTEXT01.pdf.

[28] Lennartsson, J. Data oriented interactive water: An interactive water simulation for playstation 3. Master's thesis, Linköping University, 2012. Available from: http://www.diva-portal.org/smash/get/diva2:537073/FULLTEXT01.pdf.

[29] Jakobsson, T. Parallelization of animation blending on the playstation®3. Master's thesis, Linköping University, 2012. Available from: http://www.diva-portal.org/smash/get/diva2:541451/FULLTEXT01.pdf.

[30] Gebart, J. Gpu implementation of the particle filter. Master's thesis, Linköping University, 2013. Available from: http://www.diva-portal.org/smash/get/diva2:630500/FULLTEXT01.pdf.

[31] Bond, D. A. Design and implementation of a massively multi-player online historical role-playing game. Master's thesis, Heriot Watt University, 2015. Available from: http://www.macs.hw.ac.uk/~hwloidl/publications/MScThesis_Bond.pdf.

[32] Faryabi, W. Data-oriented design approach for processor intensive games. Master's thesis, Norwegian University of Science and Technology, 2018. Available from: http://hdl.handle.net/11250/2575669.

[33] Romeo, V. Analysis of entity encoding techniques, design and implementation of a multithreaded compile-time entity-component-system c++14 library. Bachelor's thesis, University of Messina, 2016. Available from: https://www.researchgate.net/publication/305730566_Analysis_of_entity_encoding_techniques_design_and_implementation_of_a_multithreaded_compile-time_Entity-Component-System_C14_library. doi:10.13140/RG.2.1.1307.4165.

[34] Hiltunen, T. C++ robotics library. Bachelor's thesis, Kajaanin University of Applied Sci-

ences, 2018. Available from: https://www.theseus.fi/bitstream/handle/10024/144998/Teo_Hiltunen.pdf?sequence=1.

[35] Grieshofer, D. Game optimization and steam publishing for swarmlake. Bachelor's thesis, Vienna University of Technology, 2018. Available from: https://www.cg.tuwien.ac.at/research/publications/2018/GRIESHOFER-2018-GOS/GRIESHOFER-2018-GOS-thesis.pdf.

[36] Llopis, N. 2011. High-performance programming with data-oriented design. In *Game Engine Gems 2*, Lengyel, E., ed, 251–261. A K Peters.

[37] Acton, M. April 2006. Performance and good data design. https://cellperformance.beyond3d.com/articles/2006/04/performance-and-good-data-design.html. (Retrieved 01.02.2019).

[38] Acton, M. 2015. Code clinic: How to write code the compiler can actually optimize. Presented at GDC. (Retrieved 18.03.2019). URL: https://www.gdcvault.com/play/1021866/Code-Clinic-2015-How-to.

[39] Acton, M. 2018. Ecs track: Lod and culling systems that scale. Presented at Unite LA. (Retrieved 18.03.2019). URL: https://www.youtube.com/watch?v=k_ORJXmPu9M.

[40] Meagher, D. 1982. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2), 129 – 147. URL: http://www.sciencedirect.com/science/article/pii/0146664X82901046, doi:https://doi.org/10.1016/0146-664X(82)90104-6.

[41] Bentley, J. L. September 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9), 509–517. URL: http://doi.acm.org/10.1145/361002.361007, doi:10.1145/361002.361007.

[42] Acton, M. 2012. Math for game programmers. Presented at GDC. (slides only), (Retrieved 18.03.2019). URL: https://www.gdcvault.com/play/1015502/Math-for-Game.

[43] Acton, M. March 2008. Three big lies. https://cellperformance.beyond3d.com/articles/2008/03/three-big-lies.html. (Retrieved 18.03.2019).

[44] Middleditch, S. 2012. Data-oriented design. Presented at Game Engine Architecture Club at DigiPen. (Retrieved 03.02.2019). URL: https://www.youtube.com/watch?v=16ZF9XqkfRY.

[45] Nikolov, S. 2018. Data-oriented design in pratice. Presented at Meeting C++. (Retrieved 02.02.2019). URL: https://www.youtube.com/watch?v=NWMx1Q66c14.

[46] Ericson, C. 2003. Memory optimization. Presented at GDC. (Combination of audio and slides were used, available from https://www.youtube.com/watch?v=t15T_BkOtm0, (Retrieved 22.03.2019). URL: https://www.gdcvault.com/play/1022689/Memory.

[47] Ericson, C. 2004. *Real-Time Collision Detection*. CRC Press, Inc., Boca Raton, FL, USA.

[48] Acton, M. & Ericson, C. 2016. On why dod isn't a modelling approach at all. (Retrieved 02.04.2019). URL: https://sites.google.com/site/macton/home/onwhydodisntamodellingapproachatall.

[49] Proebsting, T. 1998. Proebsting's law: Compiler advances double computing power every 18 years. (Retrieved 13.05.2019). URL: http://proebsting.cs.arizona.edu/law.html.

[50] Moore, G. E. et al. 1975. Progress in digital integrated electronics. In *Electron Devices Meeting*, volume 21, 11–13.

[51] Albrecht, T. 2017. Pitfalls of object oriented programming - revisited. Presented at Tehran Game Conference 2017. (slides only), (Retrieved 08.03.2019). URL: https://docs.google.com/presentation/d/1ST3mZgxmxqlpCFkdDhtgw116MQdCr2Fax2yjd8Az6zM/edit?usp=sharing.

[52] Albrecht, T. Mar 2010. Right of reply. http://seven-degrees-of-freedom.blogspot.com/2010/?view=classic. (Retrieved 08.03.2019).

[53] Török, B. Jan 2018. Data Oriented Design with Balázs Török. Discussed on CppCast. (Retrieved 21.03.2019). URL: http://cppcast.com/2018/01/balazs-torok/.

[54] Davidović, D. May 2014. What is data-oriented game engine design? (Retrieved 22.03.2019). URL: https://gamedevelopment.tutsplus.com/articles/what-is-data-oriented-game-engine-design--cms-21052.

[55] Torvalds, L. Jul 2006. Re: Licensing and the library version of git. (Retrieved 22.03.2019). URL: https://lwn.net/Articles/193245/.

[56] Roathe, L. March 2010. Oop != classes, but may == dod. (Retrieved 22.03.2019). URL: https://roathe.wordpress.com/2010/03/22/oop-classes-but-may-dod/.

[57] Nystrom, R. 2014. *Game Programming Patterns*. Genever Benning.

[58] Twitter conversation between "@ladyaeva", Mike Acton, and Richard Fabian. (Retrieved 02.04.2019). URL: http://web.archive.org/web/20190402101726/https://twitter.com/ladyaeva/status/1093583367721967616.

[59] Janne Laine, J. Y. Data-oriented design and functional programming in unity. Bachelor's thesis, Kajaanin University of Applied Sciences, 2017. Available from: https://www.theseus.fi/bitstream/handle/10024/129313/Laine_Janne_Ylisiurua_Juho.pdf?sequence=1.

[60] Hudak, P. September 1989. Conception, evolution, and application of functional programming languages. *ACM Comput. Surv.*, 21(3), 359–411. URL: http://doi.acm.org/10.1145/72551.72554, doi:10.1145/72551.72554.

[61] Abrash, M. 1990. *Zen of assembly language: volume 1, knowledge*. Scott, Foresman & Co.

[62] Acton, M. 2019. Building a data-oriented future. Presented at WeAreDevelopers World Congress. (Retrieved 15.11.2019). URL: https://www.youtube.com/watch?v=u8B3j8rqYMw.

[63] cppreference.com. std::lock_guard. (Retrieved 15.11.2019). URL: https://en.cppreference.com/w/cpp/thread/lock_guard.

[64] Autodesk Ltd. West Riverside Hospital. (Retrieved 09.04.2019) Licensed under cc-by-sa-3.0, provided by Autodesk Inc. for research. URL: http://openifcmodel.cs.auckland.ac.nz/Model/Details/305.

[65] buildingSMART. Ifc guid. technical.buildingsmart-tech.org. (Retrieved 01.09.2019). URL: https://technical.buildingsmart.org/resources/ifcimplementationguidance/ifc-guid/.

[66] Falstein, N. 2017. A game designer's overview of the neuroscience of vr. Presented at VRDC(Virtual Reality Developers Conference). (Retrieved 09.12.2018). URL: https://www.youtube.com/watch?v=VbrM-yUkHjU.

[67] Personal communication with Vixel.

[68] Unity Technologies. Mesh. (Retrieved 04.09.2019). URL: https://docs.unity3d.com/ScriptReference/Mesh.html.

[69] Unity Technologies. Unity 2019.3 mesh api improvements. (Retrieved 08.09.2019). URL: https://docs.google.com/document/d/1I225X6jAxWN0cheDz_3gnhje3hWNMxTZq3FZQs5KqPc/edit#heading=h.vyksohcynwk5.

[70] Unity Technologies. The safety system in the c# job system. (Retrieved 08.09.2019). URL: https://docs.unity3d.com/Manual/JobSystemSafetySystem.html.

[71] Microsoft. Reference source .net framework 4.8 - list. (Retrieved 08.09.2019). URL: https://referencesource.microsoft.com/#mscorlib/system/collections/generic/list.cs.

[72] Unity Technologies. Nativedisableunsafeptrrestrictionattribute. (Retrieved 08.09.2019). URL: https://docs.unity3d.com/ScriptReference/Unity.Collections.LowLevel.Unsafe.NativeDisableUnsafePtrRestrictionAttribute.html.

[73] Unity Technologies. Mesh.getnativevertexbufferptr. (Retrieved 05.09.2019). URL: https://docs.unity3d.com/ScriptReference/Mesh.GetNativeVertexBufferPtr.html.

[74] Unity Technologies. Struct rendermesh. (Retrieved 08.09.2019). URL: https://docs.unity3d.com/Packages/com.unity.rendering.hybrid@0.0/api/Unity.Rendering.RenderMesh.html.

[75] Microsoft. Alloc(object, gchandletype). (Retrieved 08.09.2019). URL: https://docs.microsoft.com/en-us/dotnet/api/system.runtime.interopservices.gchandle?view=netframework-4.8.

[76] buildingSMART Norway. What openbim does for you - buildingsmart in four minutes. buildingsmart.org. (Retrieved 13.12.2018). URL: https://www.buildingsmart.org/about/what-is-openbim/ifc-introduction/.

[77] Eastman, C., Teicholz, P., Sacks, R., & Liston, K. 2011. *BIM handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors*. John Wiley & Sons.

[78] Statsbygg. August 2018. Digitale kontraktskrav. statsbygg.no. Retrieved 13.10.2018. URL: https://statsbygg.no/Nytt-fra-Statsbygg/Nyheter/2018/Digitale-kontraktskrav/.

[79] thebimhub.com. April 2016. Bim must become standard for construction in germany says minister. thebimhub.com. (Retrieved 13.10.2018).

[80] Department of Public Expenditure and Reform. November 2017. Government strategy to increase use of digital technology in key public works projects launched. per.gov.ie. (Retrieved 13.10.2018). URL: https://www.per.gov.ie/en/government-strategy-to-increase-use-of-digital-technology-in-key-public-works-projects-launched/.

[81] buildingSMART. Ifc overview summary. buildingsmart-tech.org. (Retrieved 18.12.2018). URL: http://www.buildingsmart-tech.org/specifications/ifc-overview.

[82] Statsbygg, S. B. 2013. manual 1.2. 1. *Statsbygg, Norway*, 809.

Acronyms

AEC	Architecture, Engineering, and Construction
API	Application Programming Interface
AoS	Array of Structures
BIM	Building Information Modelling
CAD	Computer-Aided Design
DOD	Data-Oriented Design
DOTS	Data-Oriented Technology Stack
DSMC	Direct Simulation Monte Carlo
ECS	Entity Component System
EDA	Electronic Design Automation
IFC	Industry Foundation Classes
LOD	Level Of Detail
NSD	The Norwegian Centre For Research Data
OOD	Object-Oriented Design
OOP	Object-Oriented Programming
POD	Plain Old Data
SIMD	Single Instruction Multiple Data
SoA	Structure of Arrays
VR	Virtual Reality
.dae	Digital Asset Exchange
.mtl	Material Template Library
.obj	Object Files

A Plagiarism

Comparison between the text written by Llopis[18] & Lietchy[20]. Directly identical text has been bolded (with the exception of change in punctuation and capitalization). It should be obvious to see that the parts that are not identical syntactically has almost exactly the same semantics.

Llopis:

Picture this: Toward the end of the development cycle, your game crawls, but you don't see any obvious hotspots in the profiler. The culprit? **Random memory access patterns and constant cache misses.** In an attempt to improve performance, you try to parallelize parts of the code, but it takes heroic efforts, and, in the end, you barely get much of a speed-up due to all the synchronization you had to add. To top it off, the code is so complex that fixing bugs creates more problems, and the thought of adding new features is discarded right away. Sound familiar?

Data-oriented design is a different way to approach program design that addresses all these problems. **Procedural programming focuses on procedure calls as its main element, and** OOP **deals primarily with objects.** Notice that **the main focus of** both **approaches is code:** plain **procedures (or functions) in one case, and grouped code associated with some internal state in the other.** Data-oriented design **shifts the perspective of programming from objects to the data itself: The type of** the **data, how it is laid out in memory, and how it will be read and processed** in the game.

Lietchy:

Simply using OOD practices, however, can lead to **random memory access patterns and constant cache misses. Data-Oriented Design is a different way to approach program design that addresses** these issues. **Procedural programming focuses on procedure calls as its main element, and** OOD **deals primarily with objects. The main focus of** these **approaches is code:** simply **procedures (or functions) in one case, and grouped code associated with some internal state in the other.** DOD **shifts the perspective of programming from objects to the data itself: the type of data, how it is laid out in memory, and how it will be read and processed** during program execution.

B Questionnaire

The questions that was presented to all participants. 4.1 and 4.2 were asked as follow-up questions.

1. How long have you been working with a Data-oriented Design mindset?
2. What does DoD mean for you?
3. What do you consider to be the main principles of DoD? (List and rank them according to importance)
4. What are the typical questions you ask yourself when applying DoD in practice?
    1. Do you include considerations of meta-knowledge of the data (I.e., frequency, patterns, possibilities of re-using calculations, domain specific knowledge) and its context (I.e., response deadlines, read/write access) when you ask yourself these questions?
    2. Would you consider acting on such meta-knowledge an aspect of DoD?
5. What are characteristics you expect to see in projects that correctly apply DoD?
6. What are indicators for poor application of DoD?
7. What do you consider to be the origin of Data-oriented Design?
8. What are types of problems where DoD should be applied?
9. What are types of problems where DoD shouldn't be applied?
10. What do you consider to be the main benefits of applying a DoD mindset?
11. Did certain types of bugs start to occur less frequently?
12. Did certain types of bugs start to occur more frequently?
13. Would you consider the code to be more complex than if you would have followed other processes/paradigms? (OOP, FP)
14. Can you describe productivity improvements or deteriorations that you have experienced after starting to apply DoD?
15. In your experience, what are typical issues you face when interacting with libraries where DoD isn't followed?
16. Which aspect(s) proved to be the biggest hurdle(s) you had to overcome when you started learning DoD?
17. What do you consider to be common misconceptions about DoD?
18. On a scale from 1-7 how confident are you in your understanding of Data-oriented Design? (with 1 being not confident, and 7 being very confident)
    1. Are there anything you wish to comment on regarding your confidence score?

C Use of term Data-oriented in other literature

Data-oriented design (DADES) as a system development methodology Redelinghuys, M. *A methodology for integrating legacy systems with the client/server environment*. PhD thesis, 1996

Data-oriented as encapsulation of data Romanovsky, A. & Zorzo, A. 2001. A distributed coordinated atomic action scheme. *Comput. Syst. Sci. Eng.*, 16(4), 237–247

Data-oriented as exposing of data Joshi, R. April 2007. Closer to the edge. *Electronics Systems and Software*, 5(2), 14–18. doi:10.1049/ess:20070202

Data-oriented architecture as opposed to service-oriented architecture Vorhemus, C. & Schikuta, E. 2017. A data-oriented architecture for loosely coupled real-time information systems. In *Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '17, 472–481, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/3151759.3151770, doi:10.1145/3151759.3151770

Data-oriented programming as a foundation for service-oriented architecture Joshi, R. 2007. Data-oriented architecture: A loosely-coupled real-time soa. *Whitepaper, Aug*

Data-oriented programming as languages working on semi-structured data, like XML Nawari, N. 2006. The role of cω in geo informatics systems. In *GeoCongress 2006: Geotechnical Engineering in the Information Technology Age*, 1–6

Data engineering as data sharing between different programs Wiederhold, G. & Qian, X. Data engineering. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.57.2218&rep=rep1&type=pdf

Data oriented reconfigurable models for Stream Decoders in relation to FPGA's Agosta, G., Bruschi, F., Santambrogio, M., & Sciuto, D. Sep. 2005. A data oriented approach to the design of reconfigurable stream decoders. In *3rd Workshop on Embedded Systems for Real-Time Multimedia, 2005.*, 107–112. doi:10.1109/ESTMED.2005.1518084

Data-oriented design as data dependency management matrices on GPU's Zhang, P., Gao, Y., & Qiu, M. Aug 2015. A data-oriented method for scheduling dependent tasks on high-density multi-gpu systems. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, 694–699. doi:10.1109/HPCC-CSS-ICESS.2015.314

Data-oriented architecture for stack machines Yosefpoor, A. R. & Derakhshan, P. 2013. Optimizing data-oriented architecture for generating random numbers uniform with stack machines indeterminate structure. *International Journal of Computer Science and Network Solutions*, 1(3), 26–32

Data-oriented design in AI as data mining Soler, J. *Orion, A Generic Model for Data Mining: Application to Video Games*. Theses, UBO, September 2015. URL: https://tel.archives-ouvertes.fr/tel-01201960

Data-oriented design as data delivery from a server to a mobile device Ghosh, R. K. *Data Dissemination and Broadcast Disks*, 375–407. Springer Singapore, Singapore, 2017. URL: https://doi.org/10.1007/978-981-10-3941-6_12, doi:10.1007/978-981-10-3941-6_12

Data-oriented programming as opposed to time-oriented programming on embedded platforms Vahid, F., Givargis, T., et al. 2008. Timing is everything–embedded systems demand early teaching of structured time-oriented programming. *Proceedings of WESE*, 1–9

Data-oriented design in visualization as a focus on attributes of data Romero-Gómez, R. & Díaz, P. 2016. Towards a design pattern language to assist the design of alarm visualizations for operating control systems. In *Digitally Supported Innovation*, Caporarello, L., Cesaroni, F., Giesecke, R., & Missikoff, M., eds, 249–264, Cham. Springer International Publishing

Data-oriented design as conceptual database design and object oriented design Yau, S. S. & Tsai, J. J. . June 1986. A survey of software design techniques. *IEEE Transactions on Software Engineering*, SE-12(6), 713–721. doi:10.1109/TSE.1986.6312969

Meta-data oriented design Chi-long, Z., Hong-lei, T., & Wei, S. May 2009. Integrated software engineering methodology. In *2009 International Forum on Information Technology and Applications*, volume 3, 694–697. doi:10.1109/IFITA.2009.482

Data-oriented programming in relation to the LOOPS language Brown, H., Tong, C., & Foyster, G. Dec 1983. Palladio: An exploratory environment for circuit design. *Computer*, 16(12), 41–56. doi:10.1109/MC.1983.1654267

Data-oriented programming as dataflow languages Bartenstein, T. W. & Liu, Y. D. October 2014. Rate types for stream programs. *SIGPLAN Not.*, 49(10), 213–232. URL: http://doi.acm.org/10.1145/2714064.2660225, doi:10.1145/2714064.2660225

Object oriented design as a combination of data-oriented design and operation-oriented design Schach, S. R. *Object-Oriented and Classical Software Engineering*. McGraw-Hill Pub. Co., 8th edition

Data-oriented programming where programs are "bugs" crawling around the database WEGNER, P. Dec 1976. Programming languages - the first 25 years. *IEEE Transactions on Computers*, C-25(12), 1207–1225. doi:10.1109/TC.1976.1674589

Data-oriented programming as an attack method in information security Wang, Y., Li, Q., Zhang,

P., Chen, Z., & Zhang, G. 2019. Dopdefender: An approach to thwarting data-oriented programming attacks based on a data-aware automaton. *Computers & Security*, 81, 94 – 106. URL: http://www.sciencedirect.com/science/article/pii/S016740481830659X, doi:https://doi.org/10.1016/j.cose.2018.11.002

D Interview with Dale Kim

If you feel you have already answered a question earlier, you can simply refer back to the previous answer.

1. How long have you been working with a Data-oriented Design mindset?

Trying for 7 or 8 years but only really successfully doing it for maybe 4 years.

2. What does DoD mean for you?

Having as complete of a knowledge as possible of the hardware and how the data should flow to maximize performance on that hardware. Which language you use and the details of those language are almost inconsequential so long as you have adequate access to the hardware and how memory is managed.

3. What do you consider to be the main principles of DoD? (List and rank them according to importance)

1. Understand the hardware and its capabilities.
2. Understand the data flow required to solve the problem you want to solve (and possibly rethink if the problem you have is actually the one you want to solve).
3. Set up data structures and data flow so that the final data is achieved as directly as possible (no unnecessary data transformations or excessive maintenance of state that's not relevant to the problem being solved).

4. What are the typical questions you ask yourself when applying DoD in practice?

- What are the inputs and outputs for the problem I'm looking to solve?
- Do I have more inputs than necessary to produce the outputs?
- Are all the outputs used?
- What are the dependencies of the input and output data?
- Is the problem I'm solving part of the critical path that contributes to the running time?

4.1 Do you include considerations of meta-knowledge of the data (i.e., frequency, patterns, possibility of re-using calculations) and its context (i.e., response deadlines, read/write access) when determining the inputs and outputs of a problem?

Yes, those are all included in understanding what the inputs and outputs are. Knowing those facts can allow you to make tremendous gains for performance and may often give insight into the problem that allows you to tackle it in a vastly superior or simpler way than if you did not know about those facts.

For example, a common optimization that occurs is looking at a function that is heavy on trig function evaluations. Most of the time, you have no control over the implementations of sin() or cos(); the only alternative is to write your own implementation. People are not likely to do this in order to improve the performance of this function. However, it may be the case that the vast majority of angles being passed into sin() and cos() are close to zero. If that's the case, you can approximate sin(x) by just returning x and replace cos(x) with 1. This is not always a valid optimization, but you may find this to be suitable for the problem if you know the constraints and the frequency of the data values.

4.2 Would you consider acting on such meta-knowledge an aspect of Data-oriented Design?

I wouldn't really call it meta-knowledge to begin with. Anything that you can use to your advantage to solve your problem should be utilized. That is the whole point of data oriented design, use all the data you have about the problem, your knowledge of it, the hardware, everything.

Do not guess about what happens or assume. Actually get data to back up your assumptions and drive your solutions. If you are about to embark on a performance optimization problem, don't assume you know what is slow; actually profile and get the data to tell you where the problem lies.

A higher level application of "data oriented design" in a non-performance respect would be something like coming up with a coding standard. Often this is arbitrary and comes down to the style of whoever wrote the code first. A "data oriented way" of coming up with such a standard would be to actually look at bug counts on a project and see which bugs occur most frequently and which ones could actually be addressed by writing code in specific ways that can be enforced by such a standard.

To come up with a coding standard in any other way amounts to personal preference that is not necessarily designed to achieve any outcome other than making the standard writer more emotionally pleased with the appearance of the code.

5. What are characteristics you expect to see in projects that correctly apply DoD?

Data should not be opaque and unknown. It should be very easy to determine what state is required for a certain function to run. The programmers who worked on the project should be able to answer questions about how much memory is used for a very specific piece of data in aggregate (say, all the 3D positions for static objects in a game) or understand which functions require that data.

6. What are indicators for poor application of DoD? (Elaboration added upon request)

- Random access to data.
    - Random access to data typically implies you don't know anything or enough about the

  data to structure it in the right way to maximize performance. Is the data access truly random? Usually it isn't.

- Virtual functions/abstract interfaces.
  - Again, similar to random access to data, virtual functions and abstract interfaces implies that you're going to call some code but you don't know which code you will actually call. Very rarely is it the case that you don't know what code will execute. Any time there's virtual functions, you can almost always replace it with sorted state that executes a concrete function on the whole group at the same time instead of one heterogenous array of data which calls into random functions every time.
- Heap allocations peppered throughout the code or reference counting.
  - If you heap allocate all the time in your code, it's likely that you're doing it because you don't know when you need memory and when it will go away. This implies you don't know enough about your data and must delay the decision to allocate memory until the last possible moment. Usually, you know a lot more context about your problem to be able to deal with allocating memory in a much more well defined way.

7. What do you consider to be the origin of Data-oriented Design?

I'm not sure where the true origin is, but I first became aware of it from the game development community, specifically Mike Acton.

8. What are types of problems where DoD should be applied?

I believe it can be applied to all problems. My experience thus far has shown that it's beneficial in every single place that I've attempted it. The main problem is that some problem domains have very well known implementation styles and it may be difficult to overcome the hump of solving those problems in a different way. Usually, this falls away once you truly understand the problem.

8.1. Could you give some examples of the problems and corresponding implementation styles?

GUI is notoriously object oriented. It's the way that most people learn how to do it and it is entrenched in that style. I'm not convinced anyone has really looked at that problem space and considered the design carefully for performance and ease of use.

Because of that, I think people are less willing to think that you could possibly implement GUI systems in any other way and you have to work really hard to try to get them to rethink it or you have to do it yourself and prove to people it can be done in a different way.

9. What are types of problems where DoD shouldn't be applied?

None, as far as I'm concerned.

10. What do you consider to be the main benefits of applying a DoD mindset?

Better performance and more maintainable code.

11. Did certain types of bugs start to occur less frequently?

Performance bugs are easier to deal with but I wouldn't say that a certain type of bug started to disappear drastically. Generally, it's just easier to reason about the code and data and easier to make changes that are isolated in nature to actually fix bugs.

12. Did certain types of bugs start to occur more frequently?

Again, I haven't noticed the frequency of certain classes of bugs changing, but dealing with them is much easier with DoD.

13. Would you consider the code to be more complex than if you would have followed other processes/paradigms? (OOP, FP)

Drastically simpler than OOP. Hard to compare with functional programming since I haven't written anything as extensive in a functional language, but my perception is that for me to achieve the same quality level in a functional language would require far more work.

14. Can you describe productivity improvements or deteriorations that you have experienced after starting to apply DoD?

Maintenance and bug fixing is much easier. Improving performance (which is a huge part of my job) is so much simpler that if you made other kinds of bug fixing the same, the benefit is so massive that it would still be worth doing. For example, improving performance in typical object oriented code is nearly impossible because the data arrangement and dependencies were made at too granular of a level (if at all, often that code grows organically to the point where any data design you started with was abandoned long ago). You can only do excruciatingly detailed micro optimizations which will only net you maybe 20% improvement.

A data oriented approach can allow you to scale very easily and achieve a 10x, 20x, or 30x performance improvement **and** also allow you to parallelize much more easily.

14.1 Just to clarify, I am interpreting this as: "Even if bug fixing and 'non-performance maintenance' wasn't improved, the easiness of improving performance would still make it worth it to move to DoD", is that the correct interpretation?

Yes, that's correct.

15. In your experience, what are typical issues you face when interacting with libraries where DoD isn't followed?

Being stuck with terrible performance and incomprehensible code. There are some very popular GUI libraries which have an insane amount of object orientedness that it's very difficult to understand what the structures are and how they actually map to commands the GPU will ultimately interpret.

There are also problems at the interface layer where you may need to marshall data back and forth from your format to the libraries format, but this problem is not specific to non-DoD libraries. All libraries suffer from this since you won't always have your data in the same format as the library expects.

16. Which aspect(s) proved to be the biggest hurdle(s) you had to overcome when you started learning DoD?

How to actually do it for real. The way it is described seems *magical* if you have never done it before; people claim what seems to be unreal performance gains and also purport to have large maintenance benefits. If it's so great, why doesn't everyone else do it?

The main thing is that many people think of programming in terms of what the programming language of choice lets them or makes easiest to do. This has almost no bearing on what the hardware actually wants to do. For the first few years after college, I thought I was doing "Data oriented design" because I was trying to lay out my data to account for the cache, but I was hamstrung by writing all of my code within the confines of what the C++ language design wants you to do (think writing classes with a bunch of different data members and trying to move members around based on which members are used in different functions. It simply doesn't scale).

This is data oriented design in the smallest and most basic sense because you're trying to be aware of what the hardware likes, but the benefits are so small due to trying to shove it into a language design that isn't conducive to the hardware design.

It's far simpler to look a the hardware in isolation and figure out how the data layout and flow should be for a problem. Once you figure that out, write the bare minimum code to do exactly that in whatever language of choice you have. Ignore what the language tries to make you do; implement exactly what is most beneficial and direct for the hardware to execute.

17. What do you consider to be common misconceptions about DoD?

That it takes more effort and makes software maintainability more costly. In almost every single case I've encountered, nothing could be further from the truth for a number of reasons. One big reason is that people separate performance and maintenance. Performance *is a part* of maintenance. Improving performance is vastly simpler, so much so that if everything else were the same or slightly worse, it would still be worth it in many cases.

Another reason is that testing becomes so much simpler/easier since you know what the data should be in terms of inputs and outputs that writing tests for your code becomes self evident and easy since the code doesn't have any unnecessary (or minimal) abstraction.

Part of the problem why it can be difficult to do DoD is that it's almost all or nothing. If you have a codebase that's already built in a typical object oriented way, it's challenging to take a system and "DoDify" it unless it's an isolated leaf system. And until the rest of the program is done in that way, you may not see large performance wins across the whole program until everything has been rewritten.

There are good strategies for migration, but it may not be seen as viable if you've never successfully written a well designed data oriented system.

There is also a misconception that data oriented design means you need to do everything in arrays. While it's true that very few things are more performant than an array and games care a lot about performance, it isn't a foregone conclusion that data oriented design means an implementation that's based off using lots of arrays for cache performance. Say that your primary concern isn't performance care about some other metric, like bug maintenance time. A data oriented approach would be to figure out what your main constraints are to solving your problem and how it affects bug maintenance time.

You should gather data about where all the bug fixing effort goes and study why it takes so much time for those areas of the code to be maintained. You need to think very critically about why things are costly; it's very common for people to effectively sweep things under the rug by just claiming the problem is "hard" without actually figuring out why so much development effort is put into that code. Ultimately, you may come up with an approach where the final solution isn't the highest performing implementation but is far lower maintenance cost.

17.1. For clarification, when you say performance, what do you actually mean? Game run-time performance? Developer productivity, etc, or other types?

All of it matters, but the chief concern is run time performance since that's usually the key metric that customers experience (assuming what you built actually works). Developer productivity is also a very large concern and the wins usually go hand-in-hand since the code is much easier to understand and nothing is really hidden from you.

17.2. Could you mention some of these strategies [good strategies for migration]?

There isn't one set way, it depends entirely on the problem and the existing implementation.

However, the general approach is to find parts that are easier to isolate and try to work on those first if you're pressed for time. Often, the issue is much more cancerous and you really do need to tackle the big nasty problem first, in which case you're in for a long migration period.

If you really cannot afford to rewrite a large central system in one go, then a possible approach is to find a smaller area of the system that you can rewrite and have an interop layer that basically converts the data into different formats that the different systems expect.

A concrete example: on one of my previous games, I had written everything in a C++ object hierarchy where the main game object contained a rigid body along with cached versions of physics data like radius, velocity, position, etc. I really needed to perform collision avoidance at a global level which could be processing huge amounts of data so working directly with the game object wasn't going to work.

Instead, I converted the data I needed out of the game object and stored them into arrays in SoA form and passed that data along to the collision avoidance code. This way, I could SIMD everything and consider only the data that was necessary for collision avoidance. The converting from game objects to SoA arrays, was of course expensive, but it was a one time price I paid to get everything in linear order. The speed wins I got from the SIMD collision avoidance was so massive that it was more than worth the cost. After the collision avoidance code ran, I took the results and wrote them back to the game objects.

18. On a scale from 1-7 how confident are you in your understanding of Data-oriented Design? (with 1 being not confident, and 7 being very confident)

I think 5 or 6 is a fair self assessment.

18.1 Are there anything you wish to comment on regarding your confidence score?

The biggest issue is consistently getting the best implementation possible for a particular problem at hand. It's very easy to get stuck in preconceived notions of what the problem is that could restrict you from truly high performance code. A great example I have of this recently is pathfinding.

It's often assumed that the solution to pathfinding is to create a graph and perform A* search to do pathfinding, but this is incredibly slow. For most games, you can formulate the problem in a different manner which allows you to compute all pairs shortest paths ahead of time and pathfinding becomes a trivial table lookup. There is almost zero cost to performing pathfinding in this scheme and obliterates all other search based methods in terms of runtime and often beats them in terms of memory as well.

I recently went through this experience where I tried to speed up pathfinding by improving the search, but I didn't even consider trying to precompute all the paths ahead of time since that's "obviously bad" but I never really took the time to think about what data I had. I had assumed it would be unreasonable because the number of pairs grows quadratically and I thought it wouldn't scale, but that was assuming that I did every single tile to every other tile is a different pair. In fact, you can greatly reduce the state space if you treat this problem as a visibility problem instead and deal with corners of the level rather than tiles.

Making these kinds of realizations about your data consistently and applying them to get out of the box solutions is the most challenging part of data oriented design. You often need to strip away all preconceptions of your problem and look very carefully and deeply at the data you have, the characteristics it contains and see if there are any patterns you can fully take advantage of. Often, the first goal is to simply reduce the amount of data you have (which is exactly what treating the pathfinding problem as a visiblity problem does. Instead of looking at tiles of the level, you simply look at which corners of unpathable areas you can see from any position which allows you treat huge areas of the level as one unit of data and thus reduced the state space).

Miscellaneous - Comments on DoD composability (from e-mail correspondence, included with permission):

One other thing I just realized is that OOP tends to make you see code and data as being inextricably linked and manipulated through "objects", but this leads to terrible compositional properties. I'm having difficulty coming up with a reasonably short code example right now, but I will try to explain at the high level.

Imagine you have this object hierarchy:

```
Type A --> Type B --> Type C
       \-> Type D
```

Assume Type A has a 3D position and has a position update function which can be overridden. B and C have their own implementations of the update function but C's implementation calls a non virtual function which does the real work of updating the position.

You want to implement the update function of type D but realize that type C already implemented it in its non virtual function. What do you do?

1. Move the non virtual function from type C to type A? (This is not always possible since it may be the case that type C has data that's not in type A).

2. Make a new virtual function on type A that type C overrides? (Maybe, but now you're adding overhead for declaring the new virtual function and calling it).

3. Copy/paste the implementation from type C and add it to type D?

4. Something else?

In the object oriented style, you're usually faced with making the root object deal with more and more possibilities as the hierarchy gets larger (options 1 and 2) or you decide to copy and paste your way to "victory" but now have a maintenance nightmare as you might discover a bug in the original implementation from type C and fix it but forget to make the same fix in type D (option 3).

In a typical data oriented design approach, you have data that lives separately from your code and your code has very clear inputs and outputs without any associated structure. The results come purely from feeding data into functions and you get your result there. You may have your position data laid out in memory like this:

```
Type A positions: | A0 | A1 | A2 | A3 |
Type B positions: | B0 |
Type C positions: | C0 | C1 | C2 | C3 | C4 | C5 | C6 | C7 |
Type D positions: | D0 | D1 | D3 |
void A_UpdatePosition(Vec3* positions, int count);

void B_UpdatePosition(Vec3* positions, int count);

void C_UpdatePosition(Vec3* positions, int count);
```

In this situation, it is trivial to decide that type D should use type C's implementation: you just call C_UpdatePosition(d_positions, n).

Maybe later you decide that's a bad idea and you want to use B's implementation instead, then you call B_UpdatePosition(d_positions, n).

Or maybe it really should be a totally different implementation so you go and write D_UpdatePosition().

The coupling of code and data in object oriented code makes things less composable.

E Interview with Balázs Török

If you feel you have already answered a question earlier, you can simply refer back to the previous answer.

1. How long have you been working with a Data-oriented Design mindset?

I have learned about DoD around 2010-2011 and tried to incorporate it in my work since then. It's not a mindset that I apply to everything I do though.

2. What does DoD mean for you?

It means a different approach to solving software engineering problems where the focus is on performance and creating a solution that is generally well fitted to the target architecture.

3. What do you consider to be the main principles of DoD? (List and rank them according to importance)

- All software are basically data transformation systems
- To solve a data transformation problem, you can't disregard the hardware limitations
- Data transformation is not only about raw processing power but also about accessing said data
- Data transformation can't be approached without knowing certain parameters of the data

4. What are the typical questions you ask yourself when applying DoD in practice?

- Is this part of the software a good candidate for DoD?
- What do I know about the data?
- How can I divide the data into parts so that I can avoid divergent code paths?
- How can I organize the data to provide better access patterns?

4.1. Do you include considerations of meta-knowledge of the data (I.e., frequency, patterns, possibilities of re-using calculations, domain specific knowledge) and its context (I.e., response deadlines, read/write access) when you ask yourself what you know about the data?

Definitely, when it comes to frequency and patterns. Domain specific knowledge is tricky because it can mean a lot of different things. If it means knowledge about which cases are more likely then definitely, but if it means applying some algorithmic optimizations that I don't consider part of DoD.

4.2. Would you consider acting on such meta-knowledge an aspect of DoD?

I think it is an important aspect as part of understanding the data.

4.3. Could you elaborate a bit on the criteria for what makes software a good candidate for DoD?

As I mentioned later for me DoD is not an all or nothing solution. In cases where performance and scalability is not critical but it's more important to get a solution ready as quickly as possible DoD might not be the best. Another place is where the data might be changing too fast or there's just not that much of it to make good assumptions about it.

5. What are characteristics you expect to see in projects that correctly apply DoD?

- Well thought out data layout
- Good cache utilization
- Easier multithreading
- Scalability

6. What are indicators for poor application of DoD?

- Lots of memory allocations happening all the time
- Many divergent code paths and special cases in the most frequently called functions

6.1 Could you elaborate on why you consider these to be indicators of poor application of DoD?

The first one indicates that whoever wrote that piece of the software had no idea about the data and the memory allocation cost. It's always easier to just allocate memory when we need it but it has a cost that's why preallocating is a better idea and if we have a good idea about the data we are going to work on then this is usually possible.

The second one also shows that the author didn't know the data and didn't consider the instruction cache as something important. In many cases having the data sorted or bucketed by those conditions that would cause the divergence allows to have very tight loops and good cache efficiency.

7. What do you consider to be the origin of Data-oriented Design?

I believe the origin is programmers around the world realizing that many other paradigms like OOP for example are not well suited for the actual hardware the software is supposed to run on. In terms of the games industry DoD had a big push on the PlayStation3 because the architecture was especially well suited for it.

8. What are types of problems where DoD should be applied?

Basically, any problem where high performance is crucial and a lot of data needs to be transformed, especially real time applications.

9. What are types of problems where DoD shouldn't be applied?

Problems where having a solution ready quickly is more important than having it perform well, or ones where the data or the underlying hardware are poorly specified.

10. What do you consider to be the main benefits of applying a DoD mindset?

Better understanding of the data transformation process which can result in better performance.

11. Did certain types of bugs start to occur less frequently?

I don't have any statistics to be able to reliably answer this.

12. Did certain types of bugs start to occur more frequently?

I don't have any statistics to be able to reliably answer this.

13. Would you consider the code to be more complex than if you would have followed other processes/paradigms? (OOP, FP)

It might certainly look more complex to people who are not used to this paradigm but that's probably true the other way around as well.

14. Can you describe productivity improvements or deteriorations that you have experienced after starting to apply DoD?

I don't feel like DoD provides a big productivity difference. It can certainly reduce maintenance costs in the long run, but it can also mean having to work with other programmers and explain them the concepts.

15. In your experience, what are typical issues you face when interacting with libraries where DoD isn't followed?

Since I believe that DoD can be applied in a part of the software, when interacting with the library I might not know if DoD was used somewhere or not. Usually libraries that don't apply DoD don't scale well and their performance deteriorates exponentially with more data. But even this can't be said universally because there are cases where good performance can be achieved without DoD.

16. Which aspect(s) proved to be the biggest hurdle(s) you had to overcome when you started learning DoD?

For me the biggest hurdle was getting out of the OOP mindset that university and few years of work already cemented in my mind.

17. What do you consider to be common misconceptions about DoD?

I think people treat DoD as an all or nothing solution, while in my opinion it is perfectly reasonable to apply the concepts in parts of an application while not forcing them in other parts.

18. On a scale from 1-7 how confident are you in your understanding of Data-oriented Design? (with 1 being not confident, and 7 being very confident)

5

18.1 Are there anything you wish to comment on regarding your confidence score?

[Left blank by participant]

F Interview with Marc Costa

[Marc Costa originally embedded URLs to online resources in some of his answers. For readability in a printed format these have been turned into footnotes, no footnotes was present in the original interview, so they are not marked as injected through "[]"]

If you feel you have already answered a question earlier, you can simply refer back to the previous answer.

1. How long have you been working with a Data-oriented Design mindset?

I learned about DOD around 2011 or 2012. Developing the mindset took years, though, since resources about DOD were very scarce and, after all, it's exactly this, a mindset, and not a set of design rules. I've been using a DOD mindset on personal projects for 5 years, more or less, and I've tried to apply the mindset at work for about as long. Due to other constraints (e.g. heavy OOP codebases, not being in a position to make architectural/design decisions), I haven't been able to apply the principles as broadly as I would have liked.

2. What does DoD mean for you?

DOD is a mindset that makes you look at programming from the bottom up: considering the hardware first and solving problems with that in mind. The hardware capabilities are the physical set of limitations we have to work within. Playing to the hardware strengths guarantees that the software produced will run better (e.g. better runtime performance, less memory used) than software ignoring or working against the hardware strengths.

3. What do you consider to be the main principles of DoD? (List and rank them according to importance)

1. Data is more important than code: the purpose of all code is to transform data, so the focus needs to be on the data
2. Hardware is important: the data format and the code needed to perform the transformation change depending on hardware, e.g. CPU vs GPU
3. Simplicity: write the minimum amount of code needed to do the transformation
4. Malleability: problems change, therefore data changes and the code to transform the data will change as well
5. Explicitness: define problems as data transformations with explicit inputs and outputs

4. What are the typical questions you ask yourself when applying DoD in practice?

- Do I need to do this? (can I just do nothing and get the same result from somewhere else? E.g. previous transformations)
- Can this be done earlier? Some transformations in a game engine can be avoided if e.g. the asset pipeline generates the data beforehand (need to balance memory usage & runtime cost)
- What's the exact output I want from this function / transformation?
- What's the minimum amount of code that achieves this transformation?
- Would a different data structure / data format simplify this transformation?
- What are the implications of changing the data structure / data format?

4.1. Do you include considerations of meta-knowledge of the data (I.e., frequency, patterns, possibilities of re-using calculations, domain specific knowledge) and its context (I.e., response deadlines, read/write access) when you ask yourself these questions?

Absolutely! An aspect of DOD that I didn't mention is that you should not only focus on the data transformations, but also on how common they are. That's known as the Pareto Principle or 80/20 rule: you want to focus on the code that runs 80% of the time, since optimizing the 20% doesn't have a Return On Investment as big as optimizing the 80%.

Memory vs Run-Time is also a common trade-off: depending on the time it takes to compute a result it might be worth recomputing it a few times instead of storing it, to save memory. But if it's recomputed too often it might hurt run time too much.

Domain specific knowledge is definitely taking advantage of DOD, e.g. when applying a gaussian filter to an image, you can apply a separable filter instead of the full filter for memory access and run-time performance gains.

DOD well applied is a holistic approach, meaning that the mindset needs to be applied to the entire pipeline of transformations, not just to the one right in front of us. If pre-computing some results when baking data (game editor data representation to game engine) means we can avoid a millisecond of work at run-time (and we can afford the extra memory usage), there's no point in micro-optimizing the transformation to take half a millisecond. That's still half a millisecond too much compared to the alternative.

4.2. Would you consider acting on such meta-knowledge an aspect of DoD?

Of course! See examples above.

5. What are characteristics you expect to see in projects that correctly apply DoD?

- Good performance (it doesn't mean that a game runs at target framerate during the entire development process, but that it runs at interactive framerate and that bad performance isn't hindering progress)
- Agility: new features' requirements can be easily understood and tested, and bugs fixed quickly

- Statistics / data gathering: stats on memory usage, time it takes to run each part of the code are gathered, processed and understood
- Tests: DOD makes it easier to test functional parts of the code. Note, this doesn't imply Test Driven Development

6. What are indicators for poor application of DoD?

- Poor performance
- Higher memory usage
- Complexity & rigidness

6.1 Could you elaborate on why you consider these to be indicators of poor application of DoD?

They mostly stand in contrast to what characteristics are signs of good DOD but, to elaborate:

- Poor performance: DOD is all about understanding how the underlying hardware works, so you can take advantage of that. If you work against the hardware, you end up with slow to run software. For example, in OOD, the focus is on state: objects and relationships between them. Objects tend to work very differently than how a CPU or GPU wants to process data, so there's friction between the two models that results in poorly performing code. Another example would be Functional Programming languages, the idea of pure functions (functions that don't modify state) is not a bad idea (functions with explicit inputs & outputs are generally better for understanding and optimizing), however, in FP languages all data is immutable, meaning that to modify a data structure you need to copy its entire state and add the modification to this new one. Copying large data sets is also a source of poor performance.
- Higher memory usage: the 2 previous examples also serve to explain this indicator: in FP languages, copying the data structure means we duplicate the memory the structure is using before we can free the old one. In the case of OOD, I find that since the code flow is always object to object function calls, the data flow is not as clear as when basic procedural programming is used, so temporal data tends to be stored in objects, increasing their size even though the extra memory might only be needed during a small subset of the time the object is "alive".
- Complexity & rigidness: Using OOD as an example again, when you build your model around a problem in terms of classes and relationships between them, the model is only valid for the current state of the problem. When the problem changes, there might be changes that the model supports, if you're lucky, but there might be changes that break the model entirely. Merely focusing on the data transformation, I.e. the core of the problem, and not building a model on top of that leaves the minimum amount of code that solves the problem, which makes it easier to change once the problem changes.

7. What do you consider to be the origin of Data-oriented Design?

I got introduced to DOD from blogs and publicly posted talks from people in the games industry:

- Noel Llopis blog posts: 1[1] & 2[2]
- Tony Albrecht's Pitfalls of Object-Oriented Programming (can't seem to find the original link, but there's a copy here[3])
- Mike Acton's Typical C++ Bullshit[4]
- Richard Fabian's DOD Book (at a very early stage back in 2012)[5]

The common theme behind these posts/talks is **better hardware usage**, or how to make software perform better by rearranging the data.

However, this is not new at all. Hardware performance has steadily increased a lot for a long time, but there was a time when software had to be tailored specifically to the hardware. The best example I'm aware of might be the 2 books by Michael Abrash: Zen of Assembly Language[6] and Graphics Programming Black Book[7].

With increased performance, hardware became more permissive and better at running non-optimized or badly optimized software. That's when some programming paradigms that shifted the focus from the hardware to the code itself appeared, like Object-Oriented Programming. These paradigms are what's taught in academia, specially OOP, so virtually everyone who has studied Computer Science or Software Engineering in the last couple decades knows it and writes code following Object-Oriented Design principles.

Programmers in performance sensitive areas like game developers started pushing back against using OOP as the standard design paradigm because of the performance implications it had, mainly bad memory access patterns.

This is important because, while CPUs kept getting faster, i.e. higher frequency (so more operations per second), memory latency didn't keep up (examples[8]). OOD is notoriously bad in taking advantage of memory spatial and temporal locality.

8. What are types of problems where DoD should be applied?

DOD as a mindset should always be applied, what varies, though, is the degree you can apply it. There's only so much you can accomplish with loops over arrays of inputs and outputs and,

[1] http://gamesfromwithin.com/data-oriented-design
[2] http://gamesfromwithin.com/data-oriented-design-now-and-in-the-future
[3] http://harmful.cat-v.org/software/OO_programming/_pdf/Pitfalls_of_Object_Oriented_Programming_GCAP_09.pdf
[4] https://macton.smugmug.com/Other/2008-07-15-by-Eye-Fi/n-xmKDH/i-BrHWXdJ
[5] http://www.dataorienteddesign.com/dodbook/dodmain.html
[6] http://www.jagregory.com/abrash-zen-of-asm/
[7] http://www.jagregory.com/abrash-black-book/
[8] https://duckduckgo.com/?q=processor+vs+memory+performance&atb=v134-1__&iar=images&iax=images&ia=images

as software grows, complexity arises. For example, if you want to let users of your software add features without modifying the source code, you'll need to implement an extensibility mechanism into it, probably a plug-in system. A plug-in system is inherently not very data friendly, but it can be designed in a way that the operations it allows the plugins to do, work in a DOD fashion, e.g. adding an API to integrate tasks into the job scheduler of the application.

9. What are types of problems where DoD shouldn't be applied?

See above.

10. What do you consider to be the main benefits of applying a DoD mindset?

- Simplicity: I'm a big proponent of simple solutions to problems. DOD focus on the data transformations, instead of code architecture. By shifting the focus, programmers are prevented from creating massive amounts of code in the name of some abstract goals like abstraction, clean code, generality or extensibility.

11. Did certain types of bugs start to occur less frequently?

Most of the bugs I've seen disappear are bugs that come from engineered complexity, I.e. from the code being too complex for its own good. These are inherent to OOP constructs like desynchronization in the communication between classes, hidden assumptions in class methods that the API doesn't expose, inability to separate data lifetime from object lifetime, lack of clarity of what code is multithread safe and which is not, etc...

12. Did certain types of bugs start to occur more frequently?

I haven't seen bugs specific to DOD start to occur, however some of the bugs I've always made regardless of design choice are still there: wrong loop ranges, wrong mathematical operations, wrong mapping between data formats (e.g. linear 2D array to tiled 2D array). These kind of bugs might appear more often in DOD scenarios since you're usually closer to the data representation, however they're usually quite easy to spot and fix.

13. Would you consider the code to be more complex than if you would have followed other processes/paradigms? (OOP, FP)

I find the code to be generally simpler. By focusing on the transformations alone, the code usually ends up in a straightforward layout, especially when compared to the complexity even basic OOP code adds. Of course, there are still sources of complexity, like the problem itself (inherent complexity) or the plugin system I described in #8, but even these cases benefit from simpler programming building blocks than what OOP gives you.

DOD shares some principles with FP, but FP languages usually go a lot further to keep functions pure (no data modification), e.g. making all data structures immutable. This is a source of performance issues, since copying an entire data structure to make a change is very inefficient.

14. Can you describe productivity improvements or deteriorations that you have experienced after starting to apply DoD?

DOD tends to shift the focus from state (objects) to logic (functions). I find that this helps me understand and follow the code better, therefore facilitating changing code, which is a huge productivity improvement.

Another advantage is avoiding *analysis paralysis*. This refers to the inability to make decisions when starting to write code to solve a problem due to the large space of possible solution vectors. I had this problem when trying to solve problems with OOD: what classes should I use, what should go into each class, is there a design pattern that applies here? I would also start writing classes and mock up interfaces before I wrote any code that solved the actual problem.

A possible productivity deterioration is debugging certain code snippets. Debuggers are basically state driven, I.e. when breaking into the debugger, you can inspect the current state of the program. However, due to how some memory ends up being laid out or organized, it might be hard to extract meaningful state from what the debugger gives you. Custom debugger visualizations help, but current debuggers don't make it very easy to use them.

15. In your experience, what are typical issues you face when interacting with libraries where DoD isn't followed?

There are many sorts of friction between code that follows DOD and code that is OOD:

- Memory management: most of the time, memory lifetime for the library is handled internally, tied to objects and generally they lack the possibility to specify what memory to use (instead of allocating from the standard library heap)
- Complex interactions between objects: OOD based libraries will use different objects to represent the various concepts in the library, which interact in arbitrary ways. Managing the objects then becomes the responsibility of the client.

16. Which aspect(s) proved to be the biggest hurdle(s) you had to overcome when you started learning DoD?

The biggest hurdle was letting go of OOD mindset. I had been trained on OOD for years before I even wrote one line of code on a professional setting. By then, OOD is all I knew and the only way I knew how to approach and talk about problems, so everything needed to be solved in terms of classes, objects, instances, and design patterns.

This is a big deal, since OOD encompasses the entirety of your toolset, so you have to start learning a new toolset from scratch. That's a daunting endeavor.

17. What do you consider to be common misconceptions about DoD?

The most common misconception is that DOD only applies to low-level problems or it's only useful for optimization. That commercial applications and big teams need something like OOD to organize code and share concepts.

18. On a scale from 1-7 how confident are you in your understanding of Data-oriented Design? (with 1 being not confident, and 7 being very confident)

5

18.1 Are there anything you wish to comment on regarding your confidence score?

I think I have quite a good grasp on what DOD entails and how to apply it to a variety of situations. However, I've never worked in a team where DOD was the main approach to tackle problems, so I don't have a lot of experience on professional environments centered around DOD. Having said that, I've made contributions to professional codebases following a DOD mindset, which have been successfully integrated and used by many other developers.

G Interview with Stefan Reinalter

If you feel you have already answered a question earlier, you can simply refer back to the previous answer.

1. How long have you been working with a Data-oriented Design mindset?

About 8 years.

2. What does DoD mean for you?

Writing code that is an abstraction to how data is read & written and how the data is laid out in memory (grouping by data), rather than be concerned with grouping data by objects (typical OOP encapsulation).

3. What do you consider to be the main principles of DoD? (List and rank them according to importance)

Prefer pure functions to everything else – those functions don't touch state, don't have state, and get passed all required input and output parameters.

Should be easy to parallelize, especially with pure functions (e.g. by splitting the input data set workload across several threads).

Think in terms of many, not just one. Design for the common case (many objects, large data) rather than the exception (single object).

4. What are the typical questions you ask yourself when applying DoD in practice?

How large is the data set? How often is the data touched? What is the range of the data set? Is DoD applicable to my problem?

4.1. Do you include considerations of meta-knowledge of the data (I.e., frequency, patterns, possibilities of re-using calculations, domain specific knowledge) and its context (I.e., response deadlines, read/write access) when you ask yourself these questions?

Definitely, it is very important to consider these factors to gain a better understanding of which methodology should be applied.

I'd like to give two examples:

#1: Suppose we want to build a simple GUI for our game, and we know (with reasonable certainty) that there won't be more than 100 single elements that need to be rendered each frame. In this case, I'd strive for the solution that is simplest to build, gets the job done, and is well understood by others on the team. A solution in this case could be a straightforward Widget base class, with other elements deriving from that class, implementing certain virtual functions.

For 100 elements per frame, the overhead of such an approach won't matter.

#2: Suppose we build a different game, and one of the cornerstones is complex simulation of millions of particles. Without having any extra knowledge, I know that a simple by-the-book OOP solution with virtual functions and abstractions is not going to be able to sustain a fast frame rate.

4.2. Would you consider acting on such meta-knowledge an aspect of DoD?

Yes, absolutely.

5. What are characteristics you expect to see in projects that correctly apply DoD?

Flatter class hierarchies, fewer classes, more free functions, less mutable state.

6. What are indicators for poor application of DoD?

Blindly applying DoD principles all over the place. OOP and its encapsulation have its merits and can be useful in a lot of situations.

7. What do you consider to be the origin of Data-oriented Design?

The C language.

7.1 Could you elaborate a bit on why you consider the C language to be the origin of DoD?

I guess my answer was a bit harsh and I need to elaborate.

At least from a paradigm point of view, people programming in C never had the problems we often face in big (old) C++ OOP codebases, because the language itself doesn't offer the facilities that make it so easy to corner yourself into overengineered abstractions. C doesn't offer inheritance, virtual functions, etc.

If you tell a C veteran about your newfound way of doing large transformations of data using pure functions, supplying all the input and output data without touching mutable (object/instance) state, he'd probably look at you and ask you "Of course, why would you do it any different?"

8. What are types of problems where DoD should be applied?

Transforming large sets of data, e.g. many of the operations that need to be performed in a modern video game running at 60 frames per second.

DoD should be applied when the benefits clearly outweigh the disadvantages.

8.1 Could you elaborate a bit on what you consider the disadvantages?

Not everybody on your team might be familiar with DOD, but probably everybody knows what OOP is about and how it's done in C++. DOD is probably not taught at universities/schools (yet).

Without built-in language support for things such as SoA (Structure of Arrays) or AoSoA (Arrays of Structures of Arrays), DOD tends to be a bit more complex in terms of memory allocation and deallocation.

In certain situations, the data layout used in a DOD approach might make it harder to look at the data as a whole, e.g. when stepping through code in the debugger because data is no longer encapsulated inside single instances, but can be stored in completely different data blocks. This then needs extra attention or scaffolding (e.g. debugger extensions, visualizers, plugins, etc.) in order to make the debugging experience better.

One example where this often applies is a data-oriented entity component system, where the component data itself is stored in contiguous arrays of homogeneous data, with an entity being nothing more than a few indices into those arrays. Looking at an entity instance in the debugger makes it hard to figure out *what* the data actually is.

9. What are types of problems where DoD shouldn't be applied?

If OOP leads to less code and is probably equally fast (or the performance impact really is negligible), it doesn't make sense to apply DoD to everything.

Two example:

a) A particle system transforming millions of particles every frame -> definitely a contender for DoD, otherwise performance will be less than stellar.

b) A simple GUI/widget system which will not transform more than 50 widgets per frame -> a contender for OOP, where a simple inheritance tree is probably easier to deal with.

10. What do you consider to be the main benefits of applying a DoD mindset?

You have to think about your data layout, your allocations, how the data is touched. That automatically makes it easier to write cache-friendly code and use all available cores.

Testing is a million times easier when you have pure functions. Feed them input values and you know exactly what the output values are going to be and can test for them. Compare this to OOP code that needs other class instances, global state and mock objects in order to test a single function of a single class.

11. Did certain types of bugs start to occur less frequently?

Definitely. Less worrying about "is there a data race somewhere?"„ "does this function touch global state?" and "who changed that instance's data?". Writing unit tests became much easier.

12. Did certain types of bugs start to occur more frequently?

Hard to say. DoD can make it harder to debug/diagnose certain classes of errors due to the way data may be spread into different “chunks” or classes, e.g. when considering a pure DoD-Entity-Component-System, where you no longer have a single instance of a class that can be viewed in the debugger.

13. Would you consider the code to be more complex than if you would have followed other processes/paradigms? (OOP, FP)

Depends. One definitely has to think more about the data layout upfront, but that saves time in the long run, like not having to do many of the optimization tasks that are typically done at the end of production.

When applied correctly, the code should not be more complex than the OOP counterpart – quite the opposite.

14. Can you describe productivity improvements or deteriorations that you have experienced after starting to apply DoD?

Writing unit tests was more fun, I had to spend less time on optimization (because code written with DoD in mind tends to be faster). However, I sometimes get the subjective feeling that I get less work done in the same amount of time when trying to apply DoD to a new problem (compared to a OOP solution).

15. In your experience, what are typical issues you face when interacting with libraries where DoD isn’t followed?

In my opinion, interfacing with libraries that don’t show their internals (e.g. not having source code access) is always going to be harder, no matter whether the library internally uses DoD or any other paradigm.

16. Which aspect(s) proved to be the biggest hurdle(s) you had to overcome when you started learning DoD?

Forgetting about many of the ingrained OOP paradigms, allowing myself to change my point of view and thinking.

17. What do you consider to be common misconceptions about DoD?

That DoD contradicts OOP.

18. On a scale from 1-7 how confident are you in your understanding of Data-oriented Design? (with 1 being not confident, and 7 being very confident)

5

18.1 Are there anything you wish to comment on regarding your confidence score?

No.

H Interview with Tony Albrecht

If you feel you have already answered a question earlier, you can simply refer back to the previous answer.

1. How long have you been working with a Data-oriented Design mindset?

Since about 2003.That was when I started to realise that OO generated code which ignored HW constraints, particularly memory access. Back then it wasn't called DoD, but that's when I was actively coding with data and its layout as the main considerations.

2. What does DoD mean for you?

DoD is programming by considering the flow of data through a system as the primary driving factor of program design.

3. What do you consider to be the main principles of DoD? (List and rank them according to importance)

Data coherence: keeping like data together (great for performance, great for parallelisation)

Code simplicity: code does simple things to data, transforming it in simple ways that are easily understood.

4. What are the typical questions you ask yourself when applying DoD in practice?

What is the data input, what is the output, how do I transform the input to produce the output in the simplest way possible?

4.1. Do you include considerations of meta-knowledge of the data (I.e., frequency, patterns, possibilities of re-using calculations, domain specific knowledge) and its context (I.e., response deadlines, read/write access) when you ask yourself these questions?

Definitely. The frequency and patterns of the input and output data is an important part of designing your code around your data. Understanding when you're dealing with large sets of data, or small edge cases, can help you focus where you need to be best be applying DoD. Understanding RW access can help you to make the most of the HW you're running on and produce code that works well within that framework. You want to avoid double handling your data, so reusing calculations is heavily encouraged (where possible – sometimes it is possible that precalculating data and then reusing it can be less efficient that calculating it on the fly (For example trig tables on PS3 were slower than doing the actual calculations)

4.2. Would you consider acting on such meta-knowledge an aspect of DoD?

Hmm. Good question. Hard to say. It's definitely an aspect of good, efficient programming and I would encourage a deep and wide knowledge of the system you're solving as well as the platform you're solving that system on. It is probably beyond the scope of academic DoD though.

4.2.1. For clarification, my understanding here is that you would not include acting on such meta-knowledge in a "definition of DoD" specifically, but that you do consider it a good thing in general programming. Is that correct?

Ah, ok. I see what I was getting at here. The platform here I refer to is the HW that you are running on – I was considering meta-knowledge of the platform as including knowledge of the HW – cache characteristics, R/W access speeds etc. Understanding the way your data is being used (or is to be used) **would** be part of DoD. You **do** need to understand how your data is to be used in order to apply DoD. Knowledge of the HW you are running on is necessary for efficient execution, but not for DoD practices.

5. What are characteristics you expect to see in projects that correctly apply DoD?

Simplicity and speed, data coherence

6. What are indicators for poor application of DoD?

Complexity, cache misses

6.1 Could you elaborate on why you consider these to be indicators of poor application of DoD?

One of the primary implications of DoD (for me) is performance. A primary side effect is the often complex logic required to solve problems can be dramatically simplified with DoD. If I see a complex set of classes, layers of inheritance, friend members in classes, mutables, and other OO hacks in code, I'll take a step back and look at how the dfata flows through this – what goes in, what comes out. Then, throw away all the code after the input and before the output and build something that does just what it needs to do. And only what it needs to do. Complexity often arises from solving problems that have evolved over time, or from not really understanding the basic issues at hand. Complexity in a DoD solution implies that the core problem may not be well understood – DoD applications should consist of relatively simple operations on data. Of course, those simple operations can be complex in and of themselves. Fundamentally, what I think I'm saying is that complexity is an indication of bad implementation, regardless of paradigm. All programmers should strive to produce code which is as simple as possible.

Cache misses are the antithesis of DoD – they are why DoD was first created. If you have many cache misses then your data is not being handled in a coherent manner and should be refactored.

7. What do you consider to be the origin of Data-oriented Design?

Necessity :)

It was discovered by multiple people around the same time. It was needed as our CPUs were getting faster, but the memory speed wasn't increasing at the same rate. With memory speed as the bottleneck, we had to think about optimisation and program design in a different way. Noel Llopis, Mike Acton, myself were discussing the principles of DoD in around 2007 (and probably a few others too – its been awhile. Christer Ericson used DoD principles but never really broadcast that publicly). We all discussed, blogged and spoke publicly about how we were thinking about code but the name didn't stick until I think Noel blogged about Data Oriented Design. My talk in 2009 – Pitfalls of Object Oriented Programming – was a catalyst that helped push data as a primary consideration in code design, but there were many of us working to solve the same problems in similar ways.

8. What are types of problems where DoD should be applied?

Anything that uses data. The more data you use, the more you should be applying DoD principles, particularly if performance is a consideration (which it should be).

9. What are types of problems where DoD shouldn't be applied?

You can solve any problem with a DoD mindset, but it may not be the simplest solution, initially. If you don't understand the data being used, you can't really use a DoD solution – simple procedural or OO or functional implementations can then help you to better understand your problem and the data being used – then DoD can be used. In the same way you can write an OO implementation to do anything, you can do the same for DoD

One area where DoD should be used cautiously, is when maintaining existing software. Given that DoD requires a very different mindset that OO, porting a part of a (or whole) OO system to DoD can be very time consuming and risky. Care should be taken to ensure that you're making the right choices for the right reasons.

10. What do you consider to be the main benefits of applying a DoD mindset?

The main reason for DoD, in my opinion, is to simplify your problem as much as possible, so that you are doing simple things to simple data. It produces code which is easier to maintain (as it is simple), is easier to optimise (inherently cache friendly) and is easier to parallelise (data coherency).

11. Did certain types of bugs start to occur less frequently?

[Left blank by participant]

12. Did certain types of bugs start to occur more frequently?

[Left blank by participant]

13. Would you consider the code to be more complex than if you would have followed other processes/paradigms? (OOP, FP)

No. In a lot of ways the DoD code is simpler than OOP (I've not done a lot of FP). One of the issues with OOP is the evolution of inheritance trees – they just don't scale well over time. DoD solutions scale better – you can add in new data or functionality without adversely affecting the existing data or functionality.

14. Can you describe productivity improvements or deteriorations that you have experienced after starting to apply DoD?

Usually, the immediate improvement has been performance. Once the DoD framework is in place, new code and data additions are usually simple, and the implications of changes are more easily understood.

15. In your experience, what are typical issues you face when interacting with libraries where DoD isn't followed?

Performance, complexity. Also, non-DoD interfaces can require you to change your design paradigm to fit theirs. Calling non-DoD code with DoD data can result in bad performance and complex code.

16. Which aspect(s) proved to be the biggest hurdle(s) you had to overcome when you started learning DoD?

When I started learning DoD, it didn't exist. I was learning how the hardware worked, how VLSI designers expected their HW to be used, how to take advantage of HW in the way that it was designed to be used. Initially, I didn't have the back ground in chip design, so learning that along with assembly and how compilers worked and being able to understand their output helped me to grow and understand how to write performant, data friendly code.

17. What do you consider to be common misconceptions about DoD?

That it is complex and confusing, that it should only be used for high performance programming, or, even worse, that it is unnecessary.

18. On a scale from 1-7 how confident are you in your understanding of Data-oriented Design? (with 1 being not confident, and 7 being very confident)

I'd give myself a 5.

18.1 Are there anything you wish to comment on regarding your confidence score?

There is always someone better at something than you. I know what I'm doing, but I also know that I have a lot to learn. I've not done a lot of work with applying DoD in large commercial systems, so that is an area where I could learn more.

I Interview with Richard Fabian

If you feel you have already answered a question earlier, you can simply refer back to the previous answer.

1. How long have you been working with a Data-oriented Design mindset?

When I first started in games in 2000, my boss assigned me the task of developing the animation layer of the code on a Playstation (1), and from that I developed an appreciation for how important it was to lay out your data for easy use. These were the days before caches, but the importance of data layout for saving memory and code complexity already applied. I drifted away from this mindset as I learnt more and more OO code with C++, until I started to come full circle in 2006 when I started work on the first of a series of experiments in applying database normalisation concepts to how we develop game logic. The work proved fruitful, but ultimately everything else took precedent and priorities were always on doing the necessary thing to get the game shipped. It wasn't until 2009 when I came across the term Data-Oriented Design, and had my first look at the PS3 that I realised how important some of my old work could be. In essence, I've always been thinking DOD, except for a time between 2001 and 2006 where I was an OO zealot due to overwhelming support and books suggesting it was the only good way to program software of any appreciable size. Books like the Gang of Four Design Patterns book and Code Complete were seen as important and true, and I was not experienced enough to criticise them at the time.

2. What does DoD mean for you?

DoD means thinking beyond just the algorithm, to the level of the work done by the machine. Like when you think about how architects and engineers work together, architects these days seem to design things which don't have intrinsic strong and good structure, and the engineers have to fix it. Programmers not thinking DoD, such as programmers who think in propositional logic, object-oriented, or lisp-like tree structures, often make decisions which cause massive performance costs, but also development complexity costs as they don't consider the lower level of data manipulation to bring about their design.

DoD is about using what you know, the expectations you have, and programming while considering them. For example, you cannot develop a lossy compression algorithm without DoD, as without considering the qualities of the data, then there can be no expected bias in the values and thus no "likely" sequences. DoD asks us to consider what we know, and help the computer by providing guidance, and not hiding information from it.

2.1 If I understand you and your example correctly. Given that DoD is about using what you know and your expectations, that also includes problem and domain knowledge? (not for

modeling purposes, but because there might be information there that you can take advantage of). If so, could you give some examples?

If you are looking at how you are loading data in your application, you might find there are ways in which you always do some things together. In games you might find you always load a character data file, then always load the meshes, sounds, animations, textures, and any AI scripts. The way the engine is set up, this might not be a direct link, so you can use your knowledge to decide to collect assets related not by type, but by knowing they will be loaded all at the same time and build them into a bundle of data all loaded at the same time, then handed out to each system on demand but from memory. The way a lot of PCs work, they still have spinning disks, so cutting the number of individual files per character down can save quite a bit of time, as opening and reading even the smallest of files will take as long as it takes for the head of the disk to have a chance to read the data. Remember, even the very fastest spinning disks (10k) take 6milliseconds per revolution of the disk, so on average it will take 3 milliseconds to open your file. If you have 10,000 files to load for your 100 characters with their 100 different assets each, then you are going to spend around 30 seconds just waiting to open the files. If you lump each character into little zip (not even compressed) with all their assets, then you can expect it to take 100th the time on open file.

Using our knowledge also applies in not doing work at all. Consider the simple idea of not playing very distant or quiet sounds when nearby or louder ones are playing. Without putting that idea into the code somehow, there is no way a compiler could happily decide to do that for you and be sure it is doing the right thing.

There are many many examples of this kind of thing, but people often call them premature optimisations because the act of thinking about cost seems to have been given a bad name.

A chapter missing from the DOD book release was one on counting or using human knowledge to build estimates upon which decisions about the better approach could be based. The idea is simple: a computer will act based on data, it will branch on data, but everywhere where there is a branch is a question, and what we need to do is maximise the importance of each of those branches. We do that by finding ways to not ask the trivial questions (examples of how to do that include existential processing), and we find ways of ensuring the question isn't a duplicate of a different earlier question (such as if we are doing some kind of clever symmetrical serialisation code, we don't keep asking whether we're in reading more or writing mode whenever we come up to a part where bytes are written or read).

3. What do you consider to be the main principles of DoD? (List and rank them according to importance)

Think about what you consider to be characteristic of the data you are manipulating before developing a solution.

Performance is not something to be patched in later, and is something everyone must care about, not just programmers.

Complexity and bugs come from unexpected state, so where possible, make state highly visible with paper trails where you can, and also reduce the amount of state you carry around when possible. Separate real state (some would call it the model) from the state you create to help solve the problems (some might call this viewmodel, or accidental state, or even transitional data, but it's all the same in essence, it's state which is only necessary because of how we are solving the problem and is not intrinsic to the model)

3.1 What understanding do you have in mind when you say "performance" (just run-time "speed")?

Performance is always defined in the negative. You can have a game running at 60fps, or 200fps, and they are effectively equivalent peformance because as humans, we don't care about that reduction in innefficiency. Performance must always be measured in impact on the user. Waiting an hour for a single frame to render is terrible, but waiting an hour for a home delivery is very quick. It's always going to be about expectation, but it's also not just time. Consider a game which runs at 60fps on your phone, but you notice it uses up half your battery in less than an hour. Then consider a similar game which runs at 30fps, but you play for a few hours and have only dropped 10% battery. Which would you consider to be performing better? What about two games which on average run at 30fps, except one runs at 60fps a lot of the time, and 10-15fps sometimes when things get rough. Would you consider one of them to be better performance? And then there's latency and memory usage and network tolerance and user interface responsiveness. These are all about the performance of the application. The application is performing for you, like an actor, and just as an actor can forget her lines, or not perform a regional accent well enough, it's always about how she doesn't meet expectations, not about how much she exceeds them.

4. What are the typical questions you ask yourself when applying DoD in practice?

On what data does this decision depend? (useful when debugging and when tracing memory latency bottlenecked code)

What physical limits does this hardware have? EXAMPLE: How many IOPs, mem bandwidth/latency, average seek time.

What is the minimum amount of data required to solve this problem? EXAMPLE: How much of the level geometry do I need in memory to render the first frame? Can I load in lower MIPs? Can I load in the first 20-100ms of my sounds and load the rest on demand?

5. What are characteristics you expect to see in projects that correctly apply DoD?

Regular frame times, clear allocation of duty for each thread. Data stored by how it is used. Reports on performance regularly produced from a reproducible automated process. Different code in some areas for different platforms, less expectation that all code should be platform independent. Lower energy usage and shorter build times.

6. What are indicators for poor application of DoD?

Not being able to identify any one cause of poor performance. Messy profiles with lots of bitty function calls doing things based on virtual calls or data dependent function lookups.

6.1 Could you elaborate on why you consider these to be indicators of poor application of DoD?

When you write DoD, you do not tend to write things which operate by magic (that is, not by virtual calls, or callbacks, or hierarchical fallthroughs or tree traversals, or via calls through to script which call back to code.) DoD tends to do things one chunk of work at a time. Each of these chunks are usually easier to profile, but they are also easier to reason about when it comes to the data they depend upon.

7. What do you consider to be the origin of Data-oriented Design?

Super computer technology and vector machines were the real start of it, but the origin of Data-oriented design as a term of note was the dire need of an alternative to OO programming on the in-order processors of the PS3 and Xbox 360. People united around a need to understand hardware and data, so from that point of view, I consider it to be the origin of it gaining sufficient identifiability to named.

8. What are types of problems where DoD should be applied?

When you care about producing something within the constraints of possibility, where you are likely to hit hardware and complexity limits, such as working on a non-trivial software project.

9. What are types of problems where DoD shouldn't be applied?

Prototypes, demos, toys, anything where you can throw overly powerful machines at the problem without cost to you, such as experiments where you can leave a machine running a poorly written python script overnight because developing a better solution in C++ which might run in ten minutes would take more than 24 hours to write, and there's a night's rest if you don't.

10. What do you consider to be the main benefits of applying a DoD mindset?

Not making massive mistakes with budgets of all types. When you don't think DoD, you often blindly walk into high complexity, poor memory usage, low performance solutions.

10.1. Can you exemplify what budgets you mean when you write budgets of all types?

Memory budgets (for CPU/main mem and also for GPU), loading time budgets (you will have real hardware on which the application must run, and that will have an IO throughput, so you will have a budget even if you don't know it), network bandwidth budget and error tolerance too, anything which can get too big (save game size, crash dump size, UserGeneratedContent size), or run out (file handle count, active network connections). Then there are other budgets pertaining to the development such as how long you're willing to let people develop the game without telling them if they are inside these budgets (if you don't know how long you're willing to do that, you're effectively telling them they will be cutting and redoing work at some point, but you won't tell them when.)

11. Did certain types of bugs start to occur less frequently?

Unexpected state bugs didn't seem to occur any less frequently, but they were much easier to track to cause and squash. I'd say that the number of bugs per feature stayed about the same, but debugging became easier as it was easier to tell where data was modified. *A type of bug I didn't see in DOD so much was the fake-global problem. Variables inside classes which suffer from the same issues as globals (temporal coupling and concurrency issues). In OO they were encapsulated, so couldn't be spotted. They cause lots of problems. DOD doesn't tend to have them as removing objects gives them fewer places to hide.*[1]

12. Did certain types of bugs start to occur more frequently?

In a way, yes, because development of features went quicker we saw more bugs and fixed more bugs too, but when working DoD, I definitely remember tasks being completed faster and not being returned as incomplete or unfit to use as often as when coding OO style.

13. Would you consider the code to be more complex than if you would have followed other processes/paradigms? (OOP, FP)

No, it's definitely less complex than OOP, but possibly more complex than FP, as DoD does still maintain a large amount of state compared to the pure functional languages.

14. Can you describe productivity improvements or deteriorations that you have experienced after starting to apply DoD?

When I program in DoD I tend to spend around 60-80% of my time writing features, with the remaining 20-40% split evenly between debugging and maintenance. When programming OOP, due to the difficulty in figuring out where state is changed, who changed it, when, and what state things are in, I find that maintenance and debugging take up 60-80% of my time. New features are desired, but maintaining existing code dominates the development. New code always seems to exercise code paths not usually travelled and in there we often find bugs. The DoD way tends to have simple linear transformations which either work, or don't. Introducing new code doesn't affect it in the same way.

15. In your experience, what are typical issues you face when interacting with libraries where DoD isn't followed?

Unexpected temporal coupling between API calls is the most painful. When people have tested their libraries, but only with a few sequences of operations, they often miss the way external users access their code and can cause strange bugs.

[1]The italicized text was not part of the original interview, but was added after the participant received a copy of the thesis for review. I decided not to include it in the thesis directly since it came in such a long time after the interviews were concluded, but decided to add it here for completeness.

16. Which aspect(s) proved to be the biggest hurdle(s) you had to overcome when you started learning DoD?

Other people. Many people said I was coding wrong, as if OO was somehow a universal right way to do it.

17. What do you consider to be common misconceptions about DoD?

It's all about cache-misses. DoD code is all handwritten assembly and it's unmaintainable. DoD is just applicable to old PS3 and XBOXes. DoD isn't for gameplay code.

18. On a scale from 1-7 how confident are you in your understanding of Data-oriented Design? (with 1 being not confident, and 7 being very confident)

7

18.1 Are there anything you wish to comment on regarding your confidence score?

Even though I am extremely confident I understand Data-oriented Design, I don't think I'm the best at it in the world, I know I still keep on switching into FP or OO when the design seems easier to think about from that point of view.

J Case Feedback Balázs Török

1. Do you think the proposed solution has been developed following DOD principles? Why or why not? (Please highlight potential shortcomings if existing)

To some degree. The data layout was considered when it comes to the vertex data in the voxels. There was also data that formed the basis for the disk read speed estimation. Even if this estimation turned out to be way too pessimistic it showcases a good DOD approach.

On the other hand, the layout of internal code structures wasn't described. CPU execution of the update code was assumed to be fast enough because it was based on Unity DOTS but this was not verified.

It was also not mentioned what would provide better access patterns on the GPU so I'm not sure if this was considered. The final solution contains a lot of possible data duplication in the voxels which if avoided could yield different performance characteristics both on the GPU and in the streaming system.

There is mention of avoiding too many drawcalls but there is no data about how many drawcalls would an average model contain without the optimization and how many the target platform could deal with efficiently.

2. If you believe the proposed solution does not comply with DOD principles, would the described reflections (Section 2.5.1, p. 27-31) alleviate your concern? If not, please highlight remaining concerns.

Described above

3. Is there any data or decisions that could have been considered in the process. If so, which ones?

CPU and GPU caches, memory access patterns, cost of drawcalls

4. Would you consider figure 2.1 & figure 2.2 an accurate representation of a data-oriented design principles/mindset? Why or why not?

The approach is definitely well represented. I'm not completely sure about the data categories but those are just examples in this case.

K Case Feedback Marc Costa

1. Do you think the proposed solution has been developed following DOD principles? Why or why not? (Please highlight potential shortcomings if existing)

There are DOD principles in the development of the proposed solution. However, there are also some issues, mainly with the development environment, that make a DOD solution harder or not possible at all.

These are a few instances where DOD principles are properly applied:

- Hardware constraints: the target hardware and problem specific constraints where determined early in the development (HW specs and target disk read speed).
- The design of the .str file format shows a clear DOD intent: memory mappable, sections based on the minimum amount of data needed per process, sections that are used together are close or adjacent in the file, etc...
- Pre-allocating the meshes rendering data arrays and blitting the data from the streaming file directly into the destination.
- Using Unity's DOTS and ECS to implement systems with the minimum amount of data and functionality to perform their job and being easier to perform in parallel.

There are some instances where DOD is either not properly applied or couldn't be:

- Merging vertex/index data into a single array at load time: this operation is performed repeatedly with the same inputs and outputs. This should be done offline. However, there's a tradeoff between draw call count and culling granularity. Depending on the size of the voxels a single draw call might render too much invisible geometry pushing the performance bottleneck from the CPU to the GPU.
- Unity's Mesh. When using Unity there are already many layers of abstraction between the user (i.e. the programmer in this case) and the hardware. It is very hard to calculate what the cost of these abstractions are, but it's trivial to see that it is not zero. The Mesh object is an example: how the rendering data is presented to the user might not be the final format that is sent to the GPU, certain optimizations and GPU features might or might not be exposed (e.g. mesh instancing), etc...
- Data locality in the voxel hierarchy can be improved.
- No mention of an upper limit to the amount of memory the geometry can take, the maximum number of draw calls or how to avoid low framerates when too much geometry is still on screen. Part of this is just general optimization (avoid low framerates to avoid cybersickness

in VR), but the available memory and how to make best use of it is part of a streaming system. I understand that this requirement is outside the scope of this project, though.

2. If you believe the proposed solution does not comply with DOD principles, would the described reflections (Section 2.5.1, p. 27-31) alleviate your concern? If not, please highlight remaining concerns.

Yes, mentioned improvements:

- Implementing a non-Unity IFC converter with better performance and more features (e.g. mesh classification into opaque and transparent, mesh packing and mesh splitting to optimize draw calls and voxel fitting, etc...).
- Further utilize Unity's DOTS (ECS, Jobs, new renderer). DOTS is still work in progress, so there's little one can do besides waiting for features to be released.
- Voxel hierarchy data locality.

3. Is there any data or decisions that could have been considered in the process. If so, which ones?

It's not clear from reading the document how duplicated objects are handled but, the streaming file format doesn't seem to handle them in any way. However, duplicated objects can offer some advantages:

- Memory Savings: small objects contained in multiple voxels (and/or multiple times in a voxel) could live in a separate layer in the .str file format so vertex/index data is only present once in the file.
- Duplicated geometry processing: graphics data for the duplicated objects could be preprocessed and merged into a small number of meshes created once on first load.
- Instanced Rendering: take advantage of GPU features to draw more objects with a single draw call.

Of course, it comes with tradeoffs:

- Add small objects data to the voxels: each voxel needs to keep track of mesh index and graphics data index offset for the instanced draw call.
- Different "object data types": need to differentiate between unique objects and instanced objects contained in the voxel, which will need two different processing paths (ECS jobs).

This could've fallen outside the scope of the project, but instancing has real implications on a streaming system.

4. Would you consider figure 2.1 & figure 2.2 an accurate representation of a data-oriented design principles/mindset? Why or why not?

I would not consider those figures an accurate representation of a DOD mindset. The main characteristic of a DOD mindset is that the data **transformation** is at the center of the process, which doesn't even appear on the figures.

DOD as a methodology is anti-abstraction, it focuses on data transformations by looking at the inputs and outputs and tries to streamline the transformation by exploiting properties of the data. Most of what comprises a DOD mindset is encapsulated in the **Analysis** and **Implementation** boxes.

A case specific DOD model for streaming would start by figuring out what data and which format is needed at the very end of the pipeline, e.g. list of meshes (disregard rendering for now). In your case, the transformation you start with at this point *is load entire .dae model => list of meshes.* Streaming is the attempt to streamline the transformation, i.e. load the entire .dae model, to load only what is necessary. The new pipeline becomes: *load entire .dae model => filter geometry to a subset of visible meshes => list of meshes.* The question is now: what are the inputs to **filter geometry to a subset of visible meshes**? The process continues backwards from here.

As exemplified by this simplified use case, a DOD mindset is basically the process of streamlining transformation after transformation, so the transformations are optimal, for a given definition of optimal (e.g. consume less memory, less CPU cycles, become simpler, etc...).

Clarification on memory mappable files (from e-mail correspondence, included with permission):

When I mentioned memory mappable file as an application of a DOD mindset, I was thinking about the fact that the data is directly loadable, without the need for post-processing, e.g. pointer fixups. The fact that the file is memory mappable doesn't directly imply that it needs to use the OS functions to load an entire file into memory. You could still issue specific read requests when the memory is needed as an optimization later. The point is that the format is optimized for consumption and that the reads are **optimizable** if needed.

L Case Feedback Richard Fabian

1. Do you think the proposed solution has been developed following DOD principles? Why or why not? (Please highlight potential shortcomings if existing)

Yes, but it does appear to be developed by someone inexperienced and also inexperienced in DOD, but that's good as it shows some of the pitfalls of the literature around DOD. There's not enough concentration on the basic aspects of data gathering and analysis. I had planned to add a chapter (tentatively named Counting) on how to look at data and analyse what you have and how you can turn it into usable domain knowledge. I feel a second edition will definitely have that chapter now as it could have helped you on your project.

2. If you believe the proposed solution does not comply with DOD principles, would the described reflections (Section 2.5.1, p. 27-31) alleviate your concern? If not, please highlight remaining concerns.

[Left blank by participant]

3. Is there any data or decisions that could have been considered in the process. If so, which ones?

I notice there's no reference to how Unity does its culling, and there's probably quite a few ways to make the rendering run faster if you looked into how it worked. The culling and occlusion systems in an engine often provide even more data you need to consider.

I don't understand the lure of the memory mapped solution. Personally, I would find analysing the data access to the mmap to be difficult, which hides some data from me, which feels anti DOD.

3.1. Could you elaborate on the perceived difficulty and anti-DODness of the memory mapped solution? Which alternative approach would you have chosen?

When I used mmap, it had no way of finding out or predicting when the file IO was happening nor how much data was transferred. The code I used mapped the memory segment and would load data from disk via the OS by memory faults. These were unpredictable when the data was unaligned to anything specific. I assumed your data was too. Because mmap gave little feedback as to when new memory was accessed, it meant I had to put it on a background thread. Otherwise, it caused main thread spikes when unexpected loads were triggered by memory accesses just beyond what was loaded, and these were unacceptable.

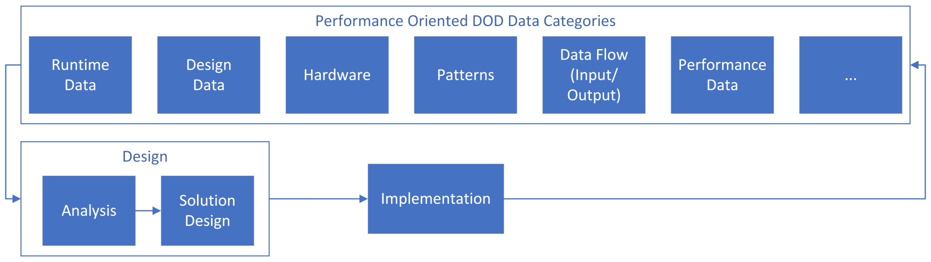
As to an alternative approach, when writing code to read parts of a file on demand, I would build an object or manager as a marshal of the data. I would use the calls into the "getter" for the data and use them to trigger disk loads and if possible, return "promises" instead of actual data. Promises, or data objects could let the client code know when the data was available and could be used to help identify the lifetime of the loaded data too. The reason this is more data-oriented, even though it sounds object-oriented, is that it's the data defining what you need to load (by identifying what data is required to satisfy the task), and by adding these accessors, you give yourself the ability to log or profile the data to determine if there are any patterns which you can use to define a better data file format.

Ultimately, the mmap version puts you in a dead-end of not being able to see the patterns of access, so you get trapped in just writing code which works but can't easily be improved. You can still make guesses, but you will have difficulty producing repeatable results of profiling and analysis. I would still use mmap if I had a lump of data I wanted to load without thinking about it, but it's a bit of a sledgehammer tool and doesn't give you enough information about what it's doing to find out if it is the cause of any issues. I've had to use platform specific profilers to both prove it was a problem and prove it wasn't a problem in different applications and their uses of the technique.

4. Would you consider figure 2.1 & figure 2.2 an accurate representation of a data-oriented design principles/mindset? Why or why not?

No, because the figures fit fine in any other development model. For them to reflect DOD, I would expect to see some runtime or data analysis picked out specifically. Without some kind of analysis of the real data flowing through the system, they're not very DOD diagrams. The way I perceive the diagrams, I feel like the Data referenced is almost entirely "design data" not actual hard data making up the meshes, or the converted binaries, or the in memory layout of the loaded str files.

4.1 Would you consider this revised figure a more accurate representation of a data-oriented design principles/mindset? Why or why not?



I would say it is neither a more accurate representation, nor less. Maybe it doesn't seem informative because the first box is a collection of aspects, and the second box is an action phase, and that box leads onto a single action. Maybe split the diagram so that actions are one box type, and concerns are represented differently?

I don't think either are clear enough to help someone who doesn't already understand what you are trying to do when designing with data in mind. Each of the boxes would need a little more information as some look like they overlap. For example, is runtime data different from performance data, if it is, then how does it differ from patterns? How is design data different from dataflow(IO)? I might add “limits” to “Hardware". Maybe the diagram simplifies the content too much. Consider writing a sentence or two on each of the aspects or categories to give the diagram more meaning.

M Extended BIM Info

M.1 What is a BIM & BIM models?

Architecture, Engineering, and Construction (AEC) projects require a lot of communication and cooperation between different disciplines to reach decisions. Historically, these decisions have been documented through written reports or architectural drawings. However, such approaches open up for version control issues, miscoordination, and miss-communication[76, 77]. Discovering such issues late in a project can have significant economic repercussions. Because of this, there has been a move towards digitization through Building Information Modeling (BIM), and several governments are encouraging or requiring its use[78, 79, 80].

A BIM model is a three-dimensional model acting as a database containing documentation surrounding project decisions, as well as information regarding specific attributes, for example, what material is used in a specific wall. Having all the data centralized in a single location minimizes issues of multiple documents with potentially conflicting information and allows project participants to quickly see where their work fits in the overarching picture[76].

M.2 IFC Format

Industry Foundation Classes (IFC) is a BIM ISO standard developed by buildingSMART[81] to ensure interoperability between different commercial BIM tools. IFC follows an object-oriented paradigm and is modeled using the EXPRESS[1] modelling language, it defines several hundreds of entities, mainly using inheritance to avoid duplication of data. IFC has quickly become the defacto industry standard, partly due to its openness, and is for instance required by the Norwegian Government[82].

[1]https://www.iso.org/standard/38047.html

N Academia Review Table

Legend:	D = Directly Mentioned	I = Implied (My interpretation)	N = Not Mentioned

	Hardware Related			Code View				Domain Knowledge			Philosophy		Patterns				Motivation, Benefits & Drawbacks			Origin	Totals		
Academia	Cache optimization	How data is organized in memory	How is the data read and processed	Program as data transformer	Focus on data rather than code	Reasoning about the code	Simplicity	Statistics in data	Acting on knowledge of the data	Data must be known before hand	Avoiding idealizing away the real world	Everything is a data problem	Relational thinking	Structure of arrays	Entity component systems	Data folding (converting bools to bitpatterns etc)	Paralleliz ation potential	Sacrificing abstraction	Highlight OOP memory shortcomings	Origin	**Direct Mention**	**Implied**	**Sum**
Liechty	D	D	D	N	D	N	N	N	N	N	N	N	N	I	N	N	N	N	D	N	**5**	**1**	**6**
Fontana et al	D	D	N	N	N	N	N	N	N	N	N	N	D	N	D (as a pattern, but conflate the two)	N	D	N	D	D (Sharp) (but also game industry)	**7**	**0**	**7**
McKendrick	D	D	I	N	N	N	N	N	N	N	N	N	N	D	N	N	N	N	D	N	**4**	**1**	**5**
Karlsson	D	D	I	D	D	N	D	N	I	D	N	N	N	D	N	N	D	N	D	I (Llopis)	**9**	**3**	**12**
Lennartsson	D	I	D	N	N	N	N	N	N	N	N	N	N	D	N	N	D	N	D	N	**5**	**1**	**6**
Jakobsson	N	N	N	N	N	N	N	N	N	N	N	N	N	D	N	N	D (SIMD)	N	N	N	**2**	**0**	**2**
Gebart	I	N	N	N	N	N	N	N	N	N	N	N	N	D	N	N	N	N	N	N	**1**	**1**	**2**
Faryabi	D	I	N	N	D	N	N	N	N	N	N	N	D	N	D (as a pattern)	N	D	N	D	N	**6**	**1**	**7**
Romeo	D	D	N	N	D	N	N	N	N	N	N	N	N	D (But not as a particular aspect of DoD)	D (as a pattern)	N	D	D (against it)	D	N	**8**	**0**	**8**
Grieshofer	D	D	N	N	N	N	D	N	I	N	N	N	D	N	N	D	D	D	D	D (Llopis)	**9**	**1**	**10**
Bond	D	D	I	N	D	N	I	N	N	N	N	N	D	N	D (yes but something different!)	N	D	N	D	N	**7**	**2**	**9**
Hiltunen	D	D	N	N	I	N	N	N	N	N	N	N	N	N	N	N	N	N	I	N	**2**	**2**	**4**
Osborn	I	N	N	N	N	N	N	N	N	N	N	N	D	N	D (pattern?)	N	N	N	N	N	**2**	**1**	**3**
Academia Count:	**12**	**10**	**5**	**1**	**6**	**0**	**3**	**0**	**2**	**1**	**0**	**0**	**5**	**7**	**5**	**1**	**8**	**2**	**10**	**3**			

O Industry Review Table

Legend: D = Directly Mentioned | I = Implied (My interpretation) | N = Not Mentioned

	Hardware Related			Code View				Domain Knowledge			Philosophy		Patterns				Motivation, Benefits & Drawbacks			Origin	Totals		
Industry	Cache optimization	How data is organized in memory	How is the data read and processed	Program as data transformer	Focus on data rather than code	Reasoning about the code	Simplicity	Statistics in data	Acting on knowledge of the data	Data must be known before hand	Avoiding idealizing away the real world	Everything is a data problem	Relational thinking	Structure of arrays	Entity component systems	Data folding (converting bools to bitpatterns etc)	Paralleliz ation potential	Sacrificing abstraction	Highlight OOP memory shortcomings	Origin	**Direct Mention**	**Implied**	**Sum**
Llopis	D	D	D	D	D	I	D	I	D	D	N	N	D	N	N	N	D (Made easier by splitting data)	N	D	N (But references Acton)	**11**	**2**	**13**
Acton	D	D	D	D	D	D	I	D	D	D (Data should be know beforehand!)	D	D	D	N	D	D	D (Made easier by splitting data)	D (Rather that abstraction s get in the way, and therefore are not "sacrificed")	I (Strongly)	N (But in opposition to C++ culture)	**16**	**2**	**18**
Fabian	D (but specifically not all DoD is about)	D	D	D	D	D	D	D	D	D (Data should be know beforehand!)	D	N	D (Suggests relational model to help separate data)	D (Can help in optimization)	D (Discussed , but from what I see, not a core concept)	D	D (DoD can help thinking in less serial manner)	D (One of the principles)	D	D (Llopis term, but has existed in various forms)	**19**	**0**	**19**
Middleditch	D	D	D	I	D	D	D	N	I	N	I	N	D	N	N	N	D	D	D	N	**10**	**3**	**13**
Nikolov	D	D	D	D	D	D	I	D	D	I (Implied by deep domain knowledge)	I	N	D	D (But only as something that works some times)	N	N	D (Made easier by splitting data)	D (Come up with better abstraction s from Domain knowledge)	D	N (But that it has been around)	**13**	**3**	**16**
Ericson	D	D	D	D	I	I	D	I	I	N	D	N	D	D	N	D	N	D	D	N	**11**	**4**	**15**
Albrecht	D	D	D	I	D	D	D	I	I	N	I	N	D	N	N	N	D	I (At least encapsulati on)	D	N	**9**	**5**	**14**
Török	D	I	I	D	D	D	N	D	D	I	N	N	I	D	D	N	N	D	I	N (Early Console Days)	**9**	**5**	**14**
Davidović	D	D	I	I	D	N	D (brute force)	I	I	N	N	N	D	D	N	N	D	I	D	N (been around for long, like Linus)	**8**	**5**	**13**
Industry Count:	**9**	**9**	**9**	**9**	**9**	**8**	**8**	**8**	**9**	**5**	**6**	**1**	**9**	**5**	**3**	**3**	**7**	**8**	**9**	**1**			

Per-Morten Straume

Investigating Data-Oriented Design