School of Economics and Management
Department of Informatics
Lund University

August 15th, 2014

Practitioners’ view on command query responsibility segregation

A thesis submitted in part fulfillment of the degree of MSc Information Systems under the supervision of Magnus Wärja

Authors: Nazife Korkmaz, Martin Nilsson
Supervisor: Magnus Wärja
Examiner: Paul Pierce

Abstract

Relational database management systems (RDBMS) have long been a predominant technology in information systems (IS). Today, however, the ever-changing technology landscape seems to be the proving grounds for many alternative approaches. For instance, alternative databases are currently used in many cloud services that affect everyday life. Similarly, a novel way to design applications has come to fruition. It relies on two concepts; command query responsibility segregation (CQRS) and event sourcing. A combination of the concepts is suggested to mitigate some performance and design issues that commonly arise in traditional information systems development (ISD). However, this particular approach hasn't sparked interest from of academia yet. This inquiry sets out to find *opportunities* and *challenges* that arise from adoption of one of the two concepts, namely CQRS. This is done in relative isolation from event sourcing. In total five interviews were conducted with seven participants using open-ended interview questions derived from design patterns research. The results are five themes that provide guidance to IS professionals evaluating adoption. These are *alignment* between IT-artifacts and business processes, *simultaneous development*, *flexibility* from specific database technology, *modularization* as a means of implementation and risk of introducing *complexity*. The results indicate that several themes from domain-driven design are influential to the concept. Additionally, results indicate that CQRS may be a precursor to eventually consistent queries and aids fine-tuning of availability, consistency and partition tolerance considerations. It is concluded that CQRS may facilitate improved collaboration and ease distribution of work. Moreover, it is hoped that the results will help to contextualize CQRS and spark additional interest in the field of IS research. The inquiry suggests further inquiries in other areas. These are among others; extract transform load-patterns, operational transforms, probabilistic bounded staleness and occasionally connected systems.

Acknowledgements

We would like to extend our gratitude to the persons that helped with this inquiry; Rinat Abdullin, Pieter Joost van de Sande, Carl Mäsak, Tore Nestenius, Tomas Roos, Jonny Svärling and Mikael Östberg.

Additional thanks go to our families and loved ones for their patience and support, especially Charlotte Rising and Orçan Uçan.

In loving memory of Britt Nilsson.

Nazife Korkmaz and Martin Nilsson, Lund 2014.

Table of Contents

1 Introduction 8
1.1 Motivation and relevance 8
1.1.1 Interesting 9
1.1.2 Applicable 9
1.1.3 Current 10
1.1.4 Accessible 10
1.2 Purpose 10
1.3 Research questions 11
1.4 Delimitations 11
1.5 Structure 12
1.6 Used expressions 12
2 Theory 13
2.1 Command query separation 13
2.2 Command query responsibility segregation 13
2.3 Event sourcing 15
2.4 Domain-driven design 16
2.5 Consistency, availability and partition tolerance theorem 17
2.6 Eventual consistency 18
2.7 Normalization and denormalization 19
2.8 Previous findings 19
3 Research framework 21
3.1.1 Architectural style and architectural patterns 21
3.1.2 Design patterns 22
3.1.3 Design research 24

3.1.4 Identified themes 24
4 Methodology 26
4.1 General approach 26
4.2 Data collection 27
4.2.1 Interview guide 27
4.2.2 Interviews 27
4.2.3 Selection of interviewees 28
4.3 Data Analysis 29
4.4 Research quality 30
4.4.1 Reliability and dependability 31
4.4.2 Validity and credibility 31
4.4.3 Ethical issues 32
5 Findings 33
5.1 Opportunities 33
5.1.1 Alignment 33
5.1.2 Simultaneous development 34
5.1.3 Flexibility 35
5.2 Challenges 36
5.2.1 Modularization 36
5.2.2 Complexity 37
6 Discussion 40
6.1 Research question 1 40
6.2 Research question 2 41
6.3 Conclusions 42
6.4 Future research 43
6.4.1 Extract, transform, load 43

6.4.2 Operational transforms 43

6.4.3 Probabilistic bounded staleness 43

6.4.4 Occasionally connected systems 43

7 Appendices 44

7.1 Interview guide 44

References 46

Table of Figures

Figure 2.1 A system with client-server architecture. It has clients, service interfaces, a model and a database. The clients connect to the service interfaces, which read the model from the database. 14

Figure 2.2 A system that untangles command form query logic. The two models reside between the services interfaces and the database 14

Figure 2.3 A model of CQRS with separated databases for commands and query models. Clients still connect to services interfaces but queries are directed towards the query database. Consequently, the databases can be managed separately. 15

Figure 2.4 A system using CQRS and event sourcing. Events are first saved in an event log that contains all events and then sent to a read database (Erb and Kargl, 2014, p. 53). 15

Figure 2.5 A sequence of five events. When the events are executed in sequence it brings the application to the final state (Erb & Kargl, 2014, p. 52) 16

Figure 2.6 An illustration of CAP is showing the partially disjoint features; consistency, availability and partition tolerance. 17

Figure 4.1 The three selection stages; experts, educators and practitioners 29

Figure 5.1 An example of an application with CQRS and CRUD deployed side-by-side as sketched by I1. Each part has a separate database and communication between is done in source code rather being distributed through the database. 37

Table of Tables

Table 1.1 Dimensions of relevance in IS research according to Benbasat & Zmud (1999, p. 13)........ 9

Table 1.2 Research questions........ 11

Table 1.3 Chapter summary........ 12

Table 1.4 Used expressions........ 12

Table 2.1 Summarization of previous findings......... 20

1 Introduction

Historically IS has often relied on the notion that an application can rely on a relational database management system (RDBMS) to manage data. RDBMSs are inherently consistent and durable, and handle operations in an atomic and isolated fashion (ACID). Businesses today sometimes run into issues balancing availability and consistency along with ACID properties. Consequently, some businesses look towards alternative solutions that offer relaxed consistency models and greater availability (Betts, Domínguez, Melnik, Simonazzi & Subramanian, 2012). This path may lead businesses towards distributed environments or cloud solutions that rely on NoSQL-databases. A novel way to design applications is to combine command query responsibility segregation and event sourcing (Erb and Kargl, 2014). Command query responsibility segregation (CQRS) is a continuation of a principle proposed by Bertrand Meyer (1988) called command query separation (CQS). Meyer (1997) argues that CQS is a profound concept that leads the way for distributed computation. Although CQS is a principle dating to the late 1980s it is today reincarnated in solutions that would have been hard to foretell decades ago.

MSDN Library recently added CQRS to its catalogue of design patterns. MSDN Library divides design patterns into eight categories; availability, data management, design and implementation, messaging, management and monitoring, performance and scalability, resiliency and security (MSDN Library, 2014). These categories come into play in enterprise systems development. Abdullin (2010) state that CQRS may aid the design of systems to cope with issues found in enterprise systems e.g. performance bottlenecks, scalability, concurrency conflicts, data staleness, complexity of the design, development and maintenance. In addition, CQRS is claimed to foster the evolution of a system and offer ways to merge conflicts at domain-level (MSDN Library, 2014).

In this thesis, we take a closer look at CQRS and related concepts in order to elaborate its implications in an information systems development (ISD) context. We believe close collaboration between the IS community, especially those devoted to ISD, and IS practitioners may aid the discovery of challenges found distributed and cloud environments. A collaborative approach to software design was advocated by the design patterns movement by pioneering authors Gamma, Helm, Johnson & Vlissides (1995) and Buschmann, Meunier, Rohnert, Sommerland & Stal (1996) but is still not commonplace two decades later (Zhang & Budgen, 2012). In fact it seems like the CQRS and event sourcing-movement is a testament to the fact that practitioners have been facing issues with system design without sparking much interest from academia.

1.1 Motivation and relevance

In order to balance research rigor against research relevance the advice put forward by Benbasat & Zmud (1999) are elaborated in this section. Benbasat & Zmud (1999) argue that IS research tends to be theory-driven rather than data-driven. This has the unfortunate side-effect that IS research may have questionable relevance in practical applications. Balancing relevance is by no means an easy feat as pointed out by Bensabat & Zmud (1999). In doing so focus shifted from reviewing current literature towards a describing a phenomena found in IS practice. Theorization is made after the commitment to a topic is established. This leads to defining desired outputs rather than reviewing possible inputs as indicated by existing literature according to Bensabat and Zmud (1999).

Table 1.1 Dimensions of relevance in IS research according to Benbasat & Zmud (1999, p. 13)

Category	Dimensions of relevance	Description
Article's content	*Interesting*	Does IS research address the problems or challenges that are of concern to IS professionals?
	Applicable	Does IS research produce the knowledge and offer prescriptions that can be utilized by practitioners?
	Current	Does IS research focus on current, at the time of publication, technologies and business issues?
Article's style	*Accessible*	Are IS research articles able to be understood (in terms of tone, style, structure, and semantics) by IS professionals? Are they written in a style that professionals would enjoy reading?

1.1.1 Interesting

CQRS has been a trending topic in the .NET-community for quite a while. It has generated a fair amount of buzz on blogs and community sites. Community experts have undoubtedly channeled a lot of interest towards this area as it is claimed to play a key role in atypical architectural approaches with remarkable features. As proclaimed by Dominguez & Melnik (2012) software development has reached a "paradigm shift" that requires the business domain to be modelled and partitioned in different ways than commonplace when relying on a RDBMS. In continuation, Dominguez & Melnik (2012) further elaborate the need for a methodology to decompose the business domain. Lately, adoption of CQRS has spread to various enterprises including financial and medical institutions (Betts et al., 2012; Malcangi, 2013). However, adoption may have come too quick for some. One of the community experts delivered the following statement in April of 2011 (Dahan, 2011, p. 1):

> "It looks like that CQRS has finally 'made it' as a full blown 'best practice'. Please accept my apologies for my part in the overly-complex software being created because of it … Most people using CQRS (and Event Sourcing too) shouldn't have done so."

As noted by Dahan (2011) indiscreet adoption of CQRS may introduce complexity and should be avoided in many cases. It seems like practitioners with past experience from CQRS and event sourcing now caution against some motives for adoption.

1.1.2 Applicable

At this point in time there are only two papers mentioning CQRS. Rajkovic, Jankovic & Milenkovic (2013) use it to write a messaging health care system but do not go into details. Erb & Kargl (2014) compare CQRS and event sourcing to a discrete event simulation (DES) and makes note of opportunities and challenges that arise from combining the concepts. A few theses at MSc-level have also emerged (Fitzgerald, 2012; Niltoft & Pochill, 2013; Hakim, 2012). Each of them illuminates various aspects of CQRS through case studies. Prior work is for the most part based on realizations of IT artifacts and the lesson learnt from those experiences. Interestingly, to this day there is no inquiry into how CQRS is perceived by experienced practitioners. Consequently, the intended use of CQRS has not yet been established and the concept cannot be readily evaluated in relation to the original intentions. As of today there is no cohesive account of what the term CQRS signifies and no academic inquiries into the opportunities and challenges that arise from adopting CQRS.

1.1.3 Current

Today some authors argue more relaxed consistency models. The extensive use of NoSQL-databases especially in cloud services (Vajk, Feher, Fekete & Charaf, 2013) often employ relaxed consistency models. Although Brewer (2001) made note of this over a decade ago, today the consequences of his consistency, availability and partition tolerance (CAP) theorem is often misunderstood or misinterpreted (Brewer, 2012). It is believed CQRS may provide details on how to practitioners work with these relaxed consistency models and how an application can be divided into consistent and eventually consistent parts. This inquiry deals with experiences of CQRS in relative isolation from eventual consistency and event sourcing, an approach that hasn't been tried before.

1.1.4 Accessible

In the post-methodology area the effectiveness of methodological rigor been extensively challenged. Many of the goals put forward by IS theorists in the past have made assumptions that have not worked out in practice (Avison & Fitzgerald, 2003). This makes it tricky to develop a research framework that is used by both practitioners and IS researchers. Consequently, outputs are inspired by Erb & Kargl's (2014) recent study that defines CQRS in terms of opportunities and challenges.

1.2 Purpose

Identifying software design processes and design patterns are generalized ways in ISD that helps to ensure more comprehensible implementation, compatible design and eases maintainability and reusability (Ampatzoglou, Frantzeskou & Stamelos, 2012). If CQRS is to be considered a design pattern as suggested by Betts et al. (2012), then contextualizing CQRS with 'experience' is one of the modes of investigation that yield the most value according to a design pattern mapping study by Zhang & Budgen (2012). A descriptive study that targets interviews with practitioner's previous work and experience with the concept complement previous case studies in the subject. The intent is to provide guidance to IS practitioners that consider adoption of CQRS by making some opportunities and challenges explicit. It is hoped that the inquiry will contextualize CQRS with emerging topics (mainly eventual consistency and event sourcing). The outcomes is targeted towards IS practitioners, especially those devoted to ISD in the field of academics and business applications. Methodological approaches, architectural models and design patterns are areas that naturally fit as topics in the IS discipline according to Vesilecas, Caplinskas, Wojtkowski, Wojtkowski, Zupancic & Wrycza (2004).

1.3 Research questions

CQRS is a concept that seems to stems from development of enterprise systems. The lessons learnt from those experiences may detail various things about emerging trends in ISD. Relating concepts found in practice to existing theories is something academia usually does pretty well. Two outcomes are emphasized to appeal equally to practitioners and IS researchers, especially those in devoted to information systems development (ISD):

- Describe opportunities and challenges that arise from adoption of CQRS
- Contextualize CQRS with eventual consistency, event sourcing, CAP theorem as well as other concepts

Erb & Kargl (2014) recently described CQRS in terms of opportunities and challenges, an approach that has inspired this inquiry. Their paper is an exceptionally clear account of CQRS in a simulation systems context. Consequently, the research questions are defined accordingly:

Table 1.2 Research questions

Question number	**Research question**
RQ_1	What is the seasoned practitioners’ view on opportunities that arise from adoption of command query responsibility segregation?
RQ_2	What is the seasoned practitioners’ view on challenges that arise from adoption of command query responsibility segregation?

1.4 Delimitations

It’s recognized that many different kinds of systems exist today, all with their own set of complexities. This account simply refers to concepts found in a business context, dealing with issues found in such environments today. Additionally, it’s also acknowledged that CQRS is an ambiguous term and may refer to an architectural style, pattern or design pattern. There will be no particular elaboration in relation to the difference in perspectives. Furthermore, it is recognized that many other concepts or patterns often come into play when describing CQRS. Those concepts will only be mentioned in limited detail. Those concepts are among others; event sourcing, eventual consistency and domain-driven design (DDD). The chosen framing assumes it can be used in various contexts, architectures and programming languages yet can be decoupled from such circumstances. Hence, CQRS is seen as an abstract, yet recurring, concept rather than a specific design or artifact. This will be elaborated later on but is an important distinction.

1.5 Structure

Researchers often come across hypothesis-testing inquiries but are strangely unfamiliar with inductive reasoning (Urquhart & Fernández, 2013). Unlike deductive reasoning, inductive reasoning lets the empirical data decide the outcome of the study. Urquhart & Fernández considers two criterions that implies the appropriateness of emergent theory; “that is fits the phenomenon and, that it helps the people in the phenomenon to make sense of their experience and to manage the phenomenon better” (Glaser, 1992 cited in Urquhart & Fernández, 2013, p. 131). Because of this the remaining chapters are summarized below.

Table 1.3 Chapter summary

Chapter	**Description**
Theory	Explores the theoretical rationale as implied by the data gathered.
Research framework	Introduces descriptive models as found in design patterns literature. This is later used to create an open-ended, semi-structured interview guide.
Methodology	Specifies details of the inquiry while emphasizing the coding practices employed.
Findings	Explores predominant themes found in the empirical data along with interpretations of these findings.
Discussion	Elaborates the finding in relation to existing theory and answers the research questions. It also concludes the thesis and suggests future research areas.

1.6 Used expressions

The table below summarizes common expressions used throughout the thesis.

Table 1.4 Used expressions

Term	**Acronym**	**Explanation**
Stale data		Stale data is another term for data that is cached or not kept up-to-date.
Atomicity, consistency, isolation, durability	ACID	Atomicity, consistency, isolation and durability-properties are typically emphasized in RDBMSs (Khashana, James & Iqbal, 2011). *Atomicity* refers to transactions being performed to completion or not at all. *Consistency* means data has been written in consistent manner once the transaction is complete. *Isolation* refers transactions only acting on committed data. *Durability* refers to data changes being permanent.
Single responsibility principle	SRP	Martin (2006) suggests that every class should have a single responsibility that is encapsulated by that class or module. The principle is generally considered a good design practice in object-oriented ISD.

2 Theory

This chapter explains CQRS as well as related concepts as indicated by the data gathered. Initially the foundational principle CQS is introduced. Then CQRS is illustrated in order to clearly convey the differences between the two concepts. Then event sourcing is inspected in more detail as it’s frequently used together with CQRS. Next, related concepts are assembled. The chapter concludes with the findings from past inquiries into CQRS.

2.1 Command query separation

CQS is about separating methods into two categories. This eases identification of two kinds of methods and exposes an effective application programming interface (Hakim, 2012). Thus, CQS is all about object-level design (methods in classes or objects). However, the concept is not new. It first appeared in 1988 when Meyer suggested that methods should be divided into two types; *functions* and *procedures*.

- *Functions* produce results yet don’t affect state.
- *Procedures* explicitly affect state but don’t provide a result.

This dichotomy is often mentioned as *commands* and *queries* and the principle *command query separation*. Both commands and queries exhibit algorithmic behavior to perform their respective tasks but are fundamentally different. A user may issue a command or ask a question, but not both at the same time. Additionally Meyer (1997, p. 751) adds that “asking a question should not change the answer”. Meyer also concludes that CQS does *not only* provide an elegant solution to distributed computation, it also prescribes how this should be done. Moreover CQS is generally accepted as an influential concept to CQRS. According to Kabbedijk, Jansen and Brinkkemper (2013) CQS and DDD are two concepts that helped to spawn the term CQRS.

2.2 Command query responsibility segregation

CQRS has been around since 2009. It is no surprise that consensus has not yet been established for the term. It’s an expression that lends itself to an abundance interpretations. Nevertheless, Fitzgerald (2012, p. 16) defines CQRS as:

> “The separation of application or system responsibilities into Writing and Reading at overall architectural level rather than internal object level”.

Clearly, Fitzgerald makes note of the association between CQRS and CQS by including it in the definition. In fact it seems CQRS is the adoption CQS at application-level design rather than object-level design. This has several implications for the architecture of an IS. To convey these implications a stereotypical example is elaborated below.

Enterprise information systems often employ client-server architecture (Betts et al., 2012; Niltoft & Pochill, 2013). Such systems usually have a top-tier containing user interface and a database tier in the other end. The middle tiers consist of business logic and models. It is not uncommon that a service interface rests on top of an aggregate with options to create, read, update and delete a record. Such services are called "'implicit' services" according to Avison & Fitzgerald (2006) though algorithmically complex services may also exist. These are services that perform complex calculations or services that respond to events by "monitoring the external environment" (Avison & Fitzgerald, 2006, p. 459). This is by no means only way to design a system, but may be regarded as a typical one.

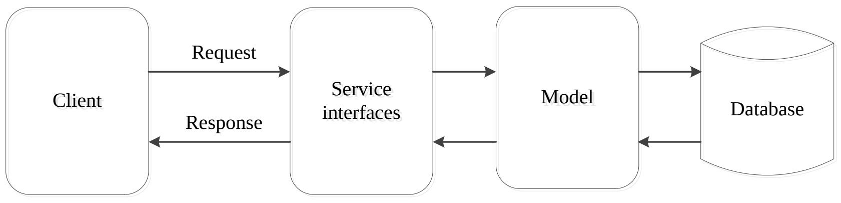

Figure 2.1 A system with client-server architecture. It has clients, service interfaces, a model and a database. The clients connect to the service interfaces, which read the model from the database.

However CQRS challenges this very notion and suggest two models; a query and a command model. This reduces operator complexity as it untangles commanding logic from query logic (Erb & Kargl, 2014). In this way, CQRS follows both the principle of CQS (Erb & Kargl, 2014), and Martin's (2006) principle of single responsibility (SRP). For CQRS, this implies one class for inserting, updating or deleting a record and another one (or several) for reading a record. This is because their responsibilities now are considered to be different (Fitzgerald, 2012).

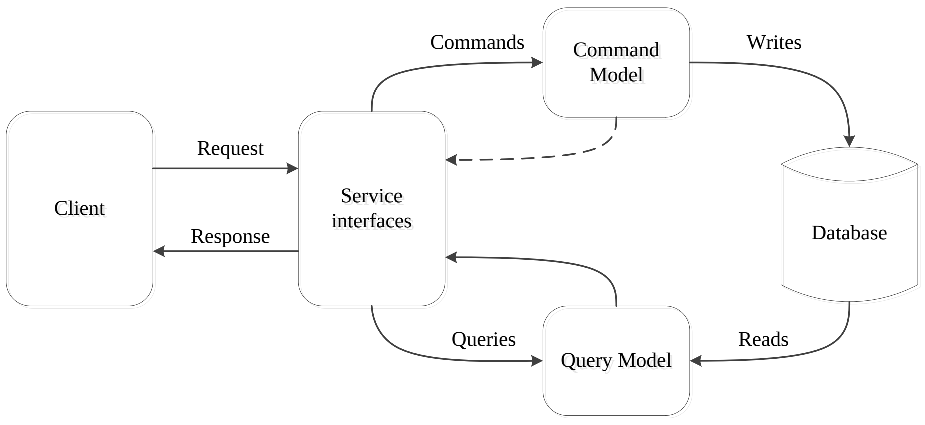

Figure 2.2 A system that untangles command form query logic. The two models reside between the services interfaces and the database

In this way CQRS introduces a separated model that distinguishes commands from queries in an application. This separation may optionally extend to the database (Erb & Kargl, 2014). This usually means that similar data now resides in two places. It also allows for two databases to used side-by-side. A RDBMS could be used for commanding while an in-memory database could be used for querying. Rajković, Janković and Milenković (2013) used such an approach and improved response time by approximately 40% for a medical IS used in the Republic of Serbia. Such designs add flexibility, and allows for databases to be managed separately (Erb & Kargl, 2014). However this approach makes it a complicated task to keep all the databases synchronized (Fitzgerald, 2012). It introduces additional considerations and certainly adds complexity (which is further elaborated in the section Eventual consistency). The term CQRS is often used analogously with such elaborate designs.

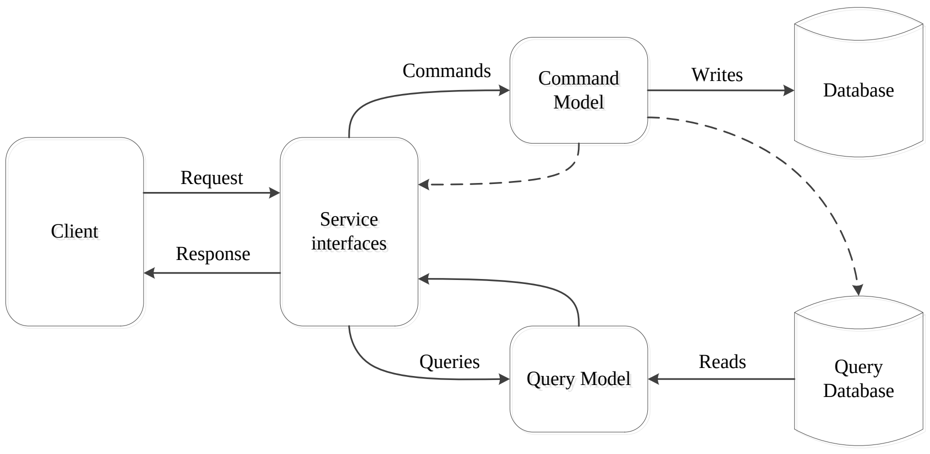

Figure 2.3 A model of CQRS with separated databases for commands and query models. Clients still connect to services interfaces but queries are directed towards the query database. Consequently, the databases can be managed separately.

2.3 Event sourcing

As mentioned previously, a novel way to design applications is to combine CQRS and event sourcing. This approach is especially common in distributed and cloud environments (Betts et al., 2012). Then, events are introduced and stored in an event log. Erb and Kargl (2014) illustrate the additions as seen in Figure 2.4.

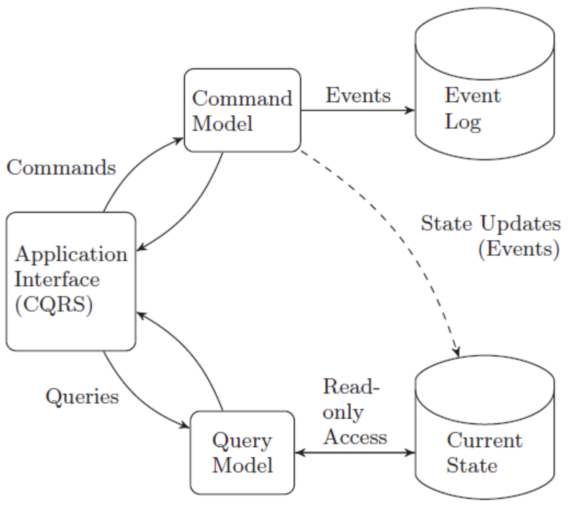

Figure 2.4 A system using CQRS and event sourcing. Events are first saved in an event log that contains all events and then sent to a read database (Erb and Kargl, 2014, p. 53).

Event sourcing implies that state is managed by adding transitions from one state to another by keeping track of the events that that took place in the system (Figure 2.5). It's an approach that keeps track of "all changes to an application state as a sequence of events" (Fowler, 2005, p. 1). Event sourcing differs from traditional means of data persistence as it doesn't manage data in tabular format (Fitzgerald, 2012). Altering values isn't done by replacing them. In this style, state is changed by adding events to the log. Consequently, the historical data is kept and must to be traversed in order to create an entity into its final state.

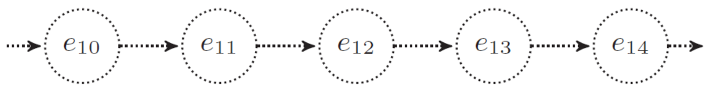

Figure 2.5 A sequence of five events. When the events are executed in sequence it brings the application to the final state (Erb & Kargl, 2014, p. 52)

It might seem like a lot of computational overhead is added, as each entity must go through all records in order to create it. But in doing so it also adds some interesting features such as the capability to add time-dependent logic (Erb & Kargl, 2014). Since the final state is dependent on traversing historical records it's not uncommon to use snapshots or in-memory databases to reduce computational overhead. Snapshots are pre-computed entities stored separately and in-memory databases usually keep the latest version of all entities in an eventually consistent manner appending events as they are propagated. Event sourcing has been suggested to be similar to discrete event simulations (DES) in that it has "an execution model represented by a sequence of events" (Erb & Kargl, 2014, p. 51).

2.4 Domain-driven design

Event souring and CQRS are closely related to the development approach domain-driven design (DDD). The approach, first described by Evans in 2004, offers techniques to cope with complex domains in business context. It emphasizes inseparable aspects of analysis and design, and collaborative aspects of ISD. Avram and Marinescu (2006) distinguishes four different elements frequently discussed in the approach; user interface, application layer, domain layer and infrastructure layer. The domain layer is seen as the heart of the business software as this is where business objects are developed and maintained (Avram & Marinescu, 2006). Some key aspects outlined by Betts et al. (2012) are:

- It encourages close collaboration between development teams and domain experts in order create a common understanding of the business domain (Erb & Kargl, 2014). The process helps to develop a *ubiquitous language* used by stakeholders and decreases the amount of translation and interpretation needed.
- The domain layer reflects business values and offer long term benefits in terms of maintaining and developing an IS. The domain objects in source code should resemblance the language used in documentation and daily work.
- The domain layer may be divided to lesser parts called *bounded contexts*. This division enables multiple teams to work concurrently and allows for multiple perspectives on similar models.

It's sometimes argued that CQRS and event sourcing can be a way to realize DDD (Truyers, 2013). Truyers (2013) claim that N-tier architectures benefit from CQRS as it helps to break down complex systems for specific requirements. In simpler terms, CQRS can be a way to succeed with DDD where typical N-tier architectures don't suffice.

2.5 Consistency, availability and partition tolerance theorem

Brewer (2012) states that systems that operate on data shared across a network have to deal with three partially disjoint features.

- *Consistency* refers to having a correct version of the data.
- *Availability* refers to the fact that every request is acknowledged with a response regardless of operational success or failure.
- *Partition tolerance* is how well a system handles unreliability, for instance network failure or incorrect messages (Gilbert and Lynch, 2012).

This is often referred to as Brewers theorem or simply CAP theorem. A common misconception is that only two of the three properties can be emphasized (Brewer, 2012). Brewer (2012) claims architectures should insist on maximizing integration of *Consistency* and *Availability*, but also considering *Partition Tolerance* where applicable. Malarvannan and Ramaswamy (2010) state that some businesses e.g. Amazon and Google use CAP theorem to evaluate distributed database systems. Modern applications need increased data and transactional fertility, which requires elastically scalable database systems (Abadi, 2012). The theorem can help to comprehend and discuss trade-offs when considering alternative storage solutions.

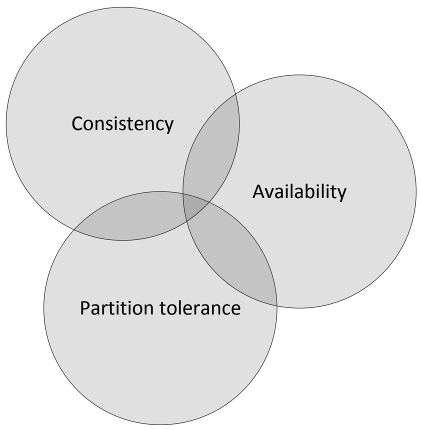

Figure 2.6 An illustration of CAP is showing the partially disjoint features; consistency, availability and partition tolerance.

Gilbert and Lynch (2012) argue that balancing the *Consistency* and *Partition Tolerance* can be done by measuring stale data. As an example an online booking system can be considered. When there are many tickets available the booking system may sell tickets while basing those requests on stale data. When the number of available tickets becomes limited then the consistency-level should be increased and any techniques for holding onto stale data should dismissed (Gilberth & Lynch, 2012). It is a complementary view that challenges the notion that *Consistency* and *Partition Tolerance* cannot be combined. Ramakrishnan (2012) claims that if a distributed system is partitioned because of failures it is not possible to guarantee both write consistency and availability while geographic replication is making all records available for reads. Hence he suggests multiple consistency models in order to align them to different situations. Niltoft & Pochill (2013) use CAP theorem to elaborate that query models may prioritize availability over consistency through eventual consistency.

2.6 Eventual consistency

Eventual consistency was noted in many years ago in distributed real-time databases (Gustavsson & Andler, 2002). Recently it has sparked the interest amongst scholars when comparing "consistent relational databases, and weakly consistent systems" (Golab, Rahman, Auyoung, Keeton & Xiaozhou, 2014, p. 40).

A simple, yet informal, definition is provided by Vogel (2009, p. 17);

> "The storage system guarantees that if no new updates are made to the object, eventually all accesses will return the last updated value"

Eventual consistency is term with many different meanings, but tends to stress the fact that data propagated across database nodes eventually will return the same value (Lloyd, Freedman, Kaminsky and Andersen, 2014; Bailis and Ghodsi, 2013). Golab et al. (2014, p. 56) hint that;

> "Recent results show that it is possible achieve the benefit of eventual consistency while providing substantially stronger guaranties, including causality and several ACID (atomicity, consistency, isolation, durability) properties from traditional database systems while still remaining highly available".

This may be one of the reasons why it is a recurring concept in cloud and distributed environments. In the context of big data and cloud solutions Wang, Sun, Deng and Huai (2013) address various issues emerged in distributed systems such as high scalability requirements, availability and complex management tasks. According to CAP theorem balancing availability and consistency is a complex feat since availability often is a requirement in distributed environments (Bailis & Ghodsi, 2013). Consequently precedence is given to availability, while consistency is relaxed through introduction of eventual consistency (Wang et al., 2013).

While CQRS relates to eventual consistency as pointed out by both Hakim (2012) and Niltoft and Pochill (2013) segregating read and write models may help to provide eventual consistency in the system, as reads and writes are managed separately. Although, eventual consistency is by no means a requirement for CQRS but rather a capability that may be used when needed (Bogard, 2012).

2.7 Normalization and denormalization

Normalization relies on rigid mathematical rationale, specifically set theory and relational algebra (Vajk, Feher, Fekete & Charaf, 2013). Its use is commonly encouraged in a RDBMS. However, alternative databases (often called NoSQL-databases) employ various forms of denormalization. Such databases affect every-day life through their use in cloud services (Vajk el al, 2013). Denormalization relies on storing pre-calculated indices or models of data. Consequently, maintaining and updating such records may become a tedious task in denormalized configurations. Both concepts deal performance issues when writing or reading from storage. Hence denormalization is not opposite of normalization, but rather a different strategy to deal with performance issues. Simply put, choosing a particular database may greatly affect whether the use of normalization or denormalization. However, when it comes to CQRS both normalization and denormalization can be used at the same time (Betts et al., 2012). Thus, it is fair to say that CQRS may rely on either strategy to resolve performance issues at hand.

2.8 Previous findings

As mentioned before, academic literature on CQRS is fairly slim. In total four publications have been found that deals with CQRS. It makes sense to summarize these accounts given the relatively unexamined state of the topic. Below follows a short summary;

- In 2012 Fitzgerald publishes an MSc thesis titled "State Machine Design, Persistence and Code Generation using a Virtual Workbench, Event Sourcing and CQRS". It breaks down the architecture of a CQRS and event sourced system. Fitzgerald also exemplifies how this can be utilized to generate code.

- "Correctness for CQRS systems" is another yet another MSc thesis. The author, Hakim (2012) intends to evaluate the correctness for eventual consistency using process meta-specification language but concludes that there are several difficulties in using this method to evaluate correctness of CQRS systems.

- Niltoft and Pochill's MSc thesis "Evaluating Command Query Responsibility Segregation" from 2013 is of great source of rationale on CQRS and provides some initial data on scalability. The authors focus on applicability of CQRS, event sourcing and DDD, and expand on CAP theorem and non-functional matters of complexity and performance.

- "Combining Discrete Event Simulations and Event Sourcing" is a paper recently published by Erb and Kargl (2014). It was presented on the International Conference on Simulation Tools and Techniques. The authors compare event sourcing to discrete event simulations and base their findings on an architectural style based on CQRS and event sourcing.

The following findings were identified through academic inquiries into CQRS. Moreover it makes it possible to compare results in terms of identifying conditions for when it becomes an appropriate concept to adopt.

Table 2.1 Summarization of previous findings.

CQRS enables read models to prioritize availability over consistency through eventually consistency.	Niltoft & Pochill, 2013
CQRS increases testability of the solution through “notable boundaries” making tests atomic and simple.	Niltoft & Pochill, 2013 (p.38)
CQRS enables simultaneous development through separation of the domain into lesser parts.	Niltoft & Pochill, 2013
CQRS is a simple concept, but DDD and Event Sourcing are complex concepts.	Niltoft & Pochill, 2013
CQRS and event sourcing enables simplified and interactive development and debugging of discrete event simulation engines.	Erb and Kargl, 2014
CQRS and event sourcing enables distributed simulation architectures	Erb and Kargl, 2014
CQRS and event sourcing enables advancements in on-line analysis and monitoring capabilities through analysis of the complete event log.	Erb and Kargl, 2014
Event sourcing is similar to state machines	Fitzgerald, 2012
Developing query models and users interfaces in isolations is simple	Fitzgerald, 2012
Code generation can be used to generate time-dependent logic for commands	Fitzgerald, 2012

3 Research framework

CQRS is a concept sometimes labeled as an architectural style (Erb & Kargl, 2014), architectural pattern or design pattern (Betts et al., 2012, Hakim, 2012). For instance, there are some indicators that should be regarded as a design pattern that denotes the separation of objects for commands and queries (Betts et al., 2012). CQRS is perhaps most frequently referred to a pattern of sorts, which also is hinted by Microsoft as it recently was added to MSDN Library's catalogue of design pattern (MSDN Library, 2014). In this chapter both architectural styles and design patterns are elaborated as two possible avenues that will aid the development of a research framework. The research framework is then later used to decompose the research questions into an open-ended, semi-structured interview guide. Exactly how this is done is elaborated in the methodology chapter.

3.1.1 Architectural style and architectural patterns

Avgeriou and Zdun (2005) state that architectural designs should be regarded as the key concept of software design. They provide solutions to architectural problems, encourage documentation of design decisions and ease communication between stakeholders. There are two branches of architectural design; *architectural style* and *architectural patterns*. They serve the same purpose on a fundamental level but some key differences are often outlined (Avgeriou and Zdun, 2005). Both approaches embrace the use of documented and proven designs to facilitate the software design process (Monroe et al., 1996). They deal with problems, technical or operational, at application-level.

Architectural styles aren't problem-solution-centric. Rather elements, relations and data flow are the detailed subjects. Attention is given to the architectural configurations, semantic of the styles, analysis that might be fulfilled on the systems constructed on the styles. They are simply interested in a "system of systems". In such a light CQRS could be one of the involved systems (Young, 2012). Architectural styles usually make note of four different elements (Monroe et al., 1996) some of which are also mentioned by Garland and Alan (1994) and Kim & Garland (2010):

- A vocabulary for elements
- Design constraints or rules that guide how organization of elements
- Semantic interpretations that have explicit meaning in the particular style
- Analyses that can be done by using the styles

On the other hand *architectural patterns* advocate design patterns' usefulness as problem and solution-elements in a context. Such patterns examine both *how* a solution helps to solve the problem and *why* it is solved. The main constituent of the approach is consequently a context-problem-solution-triplet (Avgeriou & Zdun, 2005; Buschmann et al., 1996). It may seem like CQRS fits well with the architectural patterns rationale. Betts et al. (2012) claim that the important thing concerned of such a pattern is to learn about *how*, *where* and *why* to use it. Architectural patterns commonly deal with issues like scalability, complexity and managing business rules.

Architectural patterns and architectural styles ideas helps to frame CQRS at a general level. But in continuation design patterns are also considered in order to create a research framework for the thesis. In addition there are plenty of quotes framing CQRS as a design pattern though there is a bit of a debate on the subject. Nevertheless, current literature on architectural patterns, architectural styles and design patterns had a tremendous impact in shaping the research strategy and defining appropriate inputs and outputs.

3.1.2 Design patterns

The architect Christopher Alexander (1977, p. x) describes patterns as:

> “Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without doing ever doing it the same way twice.”

Elaborating on this explanation Alexander refers to abstract design rather than concrete design. The distinction is subtle at first glance, but has tremendous impact on how design knowledge is externalized and abstracted away from some issues but not others. Having originated from architectural theory the concept today extends to a many different disciplines. It has previously been studied in Informatics (Ampatzoglou, Stamelos & Frantzeskou, 2012), Informatics with Applications in Biomedicine (Fountoukis & Chatzistavrou, 2012) and Computer Sciences to name a few areas. A common denominator for many such studies is the extensive reference to the patterns described by the pioneering authors Gamma et al. (1995).

Gamma et al. (1995) argues that design patterns help us to learn, reuse and refactor parts of object-oriented designs. Patterns are basic building blocks that solve particular issues while catering for the overall architecture. They are lesser in reach than a system or library architecture but can shape such architectures in various ways. The relation between design patterns and architecture is a complex one. While design patterns are used to help solve specific problems they still need to make sense in the parenting architecture (Gamma et al., 1995). Buschmann et al. (1996), however, does not make this distinction. He argues that patterns may describe an architecture as well. In short design patters can elaborate designs on different levels of abstraction. It may refer to organization of classes and object (Gamma, 1995), or the interplay of different structures (Buschmann et al., 1996; Jain & Kircher, 2004). Clearly opinions differ in this respect. Meyer (1997) asserts that most important task is to identify and name design patterns. Once this is done knowledge can be attached to it and then they can be elaborated, discussed, learnt and refined, and take on a life of their own. Patterns offer guidance to practitioners how these issues can be solved and how they relate to other patterns. However patterns are not alone in doing so, concrete examples can also provide some of the same benefits (Meyer, 1997).

Critics often contest the lack of grounded theory in the subject. One critic even goes as far as claiming that the patterns movement is “an eclectic practitioners approach” (Wendorff, 2001, p. 79). It’s a subject that needs a lot of work, no doubt. The evidence-based empirical research stretches pretty thin (Zhang & Budgen, 2012). One has to consider that design patterns deal with “ill-structured” issues and are hard to solve systematically (Zhang & Budgen, 2012).

To describe design patterns Gamma et al. (1995) suggest a template consisting of 13 topics focusing on intent, motivation, applicability and consequences. These topics serve the purpose of clarifying and resemble commonly occurring themes as commonly done in the rational design process elaborated by Simon (1996). The mentioned topics are presented below which were utilized in order to decomposing interview questions into subgroups. The template also aided analysis of the data.

Pattern Name and Classification
The name and type should be defined clearly.

Intent
- What does the design pattern do?
- What is its rationale and intent?
- What particular design issue or problem does it address?

Also Known As
Synonyms used to denote the same pattern.

Motivation
An illustration of the design problem through a scenario often contains classes and objects structures.

Applicability
The following questions define applicability:
- What are the situations in which the design pattern can be applied?
- What are examples of poor designs that the pattern can address?
- How can one recognize these situations?

Structure
The collaboration between objects should be modelled if possible with class and interaction diagrams.

Participants
Detail the participants of the design patterns are classes and/or objects and include their responsibilities.

Collaborations
This is related with how the participants cooperate to handle their responsibilities.

Consequences
Presenting the results is crucial in terms of benefit from the pattern, under the provided questions:
- How does the pattern support its objectives?
- What are the trade-offs and results of using the pattern?
- What does the design pattern objectify?
- What aspect of system structure does it allow to be varied independently?

Implementation
Are there remarkable issues that need to be emphasized?
- What pitfalls, hints, or techniques should one be aware of when implementing the pattern?
- Are there language-specific issues?

Sample Code
Code pieces could be presented that how in fact it might be implemented in a suitable programming language.

Known Uses
Some examples should be provided from a real system.

Related Patterns
Addressing the related patterns is important in terms of evoking new inventions.

- What design patterns are closely related to this one?
- What are the important differences?
- With which other patterns should this one be used?

3.1.3 Design research

Design science in information systems (IS) basically refers to build a software artifact that solves a human problem and this artefact is examined in terms of solving problem and its efficiency (Hevner & Chatterjee, 2010). More specifically, it proposes innovations which helps to describe the practices, ideas, technical capabilities and products and in IS research these could be efficiently achieved (Hevner, March, Park & Ram, 2004). Markus et al. (2002) specifies some theories that are defined as integrated artefacts of IS design science research such as; a particular class of user requirements, effective development practices and a type of system solution. The evaluation of such theories could provide benefits for specific problems (Hevner et al., 2004). In the light of mentioned arguments, since CQRS has been already defined as a design pattern by some of its proponents, the further investigation was conducted by applying qualitative methods in order to yield improved evaluation of designed artefacts (Hevner et al., 2004).

Vesilecas, Caplinskas, Wojtkowski W., Wojtkowski W.G., Zupancic and Wrycza (2004) state according to March and Smith (1995) that there are four research activities for design and natural science: build, evaluate, theorize and justify. Besides the authors claim that building and evaluation address design science activities while justification and theorizing referring natural science activities. In that sense evaluation as a process helps to identify if there is an improvement in relation with the performance of existing artifacts. Similar to evolving of CQRS most of the IT artifacts are developed usually by a designer they specified. However the emergent features of the artefacts cannot be inferred from previous designs. This designates that permanent redevelopment is required. This inquiry intends to explore *why* and *when* it should be considered as a part of an ISD process from practitioners' point of view.

3.1.4 Identified themes

Erb and Kargl's (2014) paper focuses on event sourcing and other CQRS-related concepts. Similar to the Erb and Kargl paper opportunities and challenges are sought out. But in this case CQRS is the topic of interest. The main characteristics of the concept are investigated through interpretation of design patterns template provided by Gamma et al. (1995). The template helps to establish what empirical material to be gathered. This aids the process of both developing and decomposing questions in terms of finding answers to the research questions by allocating the topics or grouping topics according to the template.

Opportunities refer to benefits and estimated benefits of applying CQRS. Moreover, the related concepts are used to contextualize CQRS. Intent, applicability and related patterns topics are the most utilized ones from the template.

Challenges were sought from applicability, consequences and implementation areas of the template. This identifies potential and current risks and liabilities and consequently challenges.

4 Methodology

In this chapter we present our research approach and explain these in order to motivate the chosen methods. Afterwards we give details about the data collection processes of the thesis by utilizing literature of methodologies. Data collection stage is introduced carefully since it is a core activity. Then the analysis is described in the light of coding structure put forward by Broom (2005). Finally, research quality in qualitative researches is elaborated. This guides softer quality aspects of the study.

4.1 General approach

Qualitative approaches are used to help researchers understand a phenomenon in its natural context while the quantitative approaches focus on measurement (Recker, 2013). Qualitative descriptive methods yields information about object in its natural setting in a language that is common to the target audience (Sandelowski, 2000). The approach used in this inquiry is initiated by investigation of phenomena, and theorizing and coding is done in parallel. This collaborates with qualitative descriptive methods and relevance aspects as mention by Bensabat and Zmud (1999). Within these methods an exploratory approach was chosen since case studies has already been done in the area (Niltoft & Pochill, 2013; Fitzgerald, 2012). According to Parse (2001) these are the two approaches available within qualitative descriptive methods. Qualitative methods assist in expressing participants’ views on the subject. It also attracts the attention to processes, patterns and structural features (Flick, Kardoff & Steinke, 2004). Accordingly to the main purpose of this inquiry, qualitative methods seemed to fit reasonably well.

There are several techniques could be applied in qualitative research when it comes to data collection stage. Interviews are the most frequently used methods one. Interviews include among others face-to-face, telephone or focus groups conversations. Observational techniques and documentation can also be used (Bhattacherjee, 2012). Qualitative interviews are about understanding the world from the subject and proponents point of view by exposing the attributes of their lived world (Kvale, 2006). Furthermore, strength of interviews is that they can deliver specific information in a few seconds, a feat hard to accomplish by observational work (Seale, 1999b). In this inquiry written documents, blogs posts, online resources and source code was considered since they contain valuable information provided by experts in the area. However, the reciprocal nature of interviews was regarded as imperative to gain an understanding of the topic. Interviews will negotiate language between the participants of the interview. Interpretation of documents in an observational work would raise reliability concerns due to the technical nature of those documents. Qualitative interviews allow questions to be elaborated, explained in more simple terms. Consequently, interviews were adopted as the primary data-collection technique.

4.2 Data collection

4.2.1 Interview guide

Doody and Noonan (2013) state interviews benefit from good interaction with the interviewees. The first minutes of the interviews' are crucial since this is when the participants get acquainted before sharing their experiences and thoughts (Kvale & Brinkmann, 2009). Therefore the interview guide was prepared with some general questions letting the interviewer and interviewee's present themselves. This was believed to help establish a friendly atmosphere before talking about topic of interest. This is reflected in the interview guide as it begins with some introductory questions. During the first couple of interviews and conversations leading up to interviews it was noted that the intention of the study was a major motivating factor from the interviewees' point of view and it was frequently asked for and elaborated. Consequently separate part pertaining only to the purpose of the study was eventually added to the interview guide.

Gamma's et al. (1995) template for structural patterns evaluated against other templates (Buschmann et al., 1996; Simon 1996; Jain & Kircher, 2004) and formal style (Fowler, 2003) and architectural styles (Monroe, 1996) to describe CQRS. It was decided that Gamma's et al. (1995) template was the most suitable one, though difference between the different design patterns templates were minor. Design patterns are believed to have both academic relevance and practical applicability as it is a tool used in both areas. Additionally, the design patterns approach is arguably a more developed concept than architectural styles. Also, design patterns is arguable a familiar concept for most practitioners. Naturally, simple binary questions were omitted. Some questions were rephrased or inversed to ensure richness in the responses received. Thus, it ensured that questions would give responses that would be of value or could be elaborated during the interview.

The interview guide concludes with some closing questions that evaluate relevance of questions asked and additional refinement of the questions. This resulted in some minor adjustments of the questions during the interviews, e.g. the ordering of questions and the phrases used. Additionally, it also helped to evaluate construct validity at an early stage. The guide concludes with an explanation of public availability of the material and a question regarding participatory anonymity. In summary, the interview guide is divided into the following five sections (available in complete form in appendix 1);

- The purpose of the study
- General questions
- CQRS questions
- Closing questions
- Debrief

4.2.2 Interviews

Semi-structured interviews were conducted in this study that we could use open-ended questions in order to give chance exploring the concept from different aspects as interviewees willing to share (Longhurst, 2009). The main motive for using semi-structured interviews is to expose specific issues previously not realized which could be discussed or could be examined empirically (Horton, Macve & Struyven, 2004). Trying to describe the CQRS through interviews seemed beneficial as it would yield comprehensive results and make it possible to deal with uncertainties by asking more questions during the interview. Some problems that might occur when adopting qualitative interviews such as time limitations, ambiguity in language and lack of trust (Myers & Newman, 2007). If such issues arise it may affect completeness of the collected data. As a precautionary measure interviews were targeted to be roughly an hour. The time-constraints were found to be quite appropriate, even though some conversations lasted substantially longer.

After each interview transcriptions was immediately started. Memos and notes regarding consent or confidentiality gathered and organized. This also helped to evaluate progress and necessity of additional interviews. During the first interview, it was realized that a couple of questions might have been considered leading, as some answers came prior to the question being asked. As a precautionary measure a few questions were rephrased.

Doody and Noonan (2013) compare two ways of recording the interviews; writing notes and audio recordings. The inherent risk of missing out on details it was decided that recordings should be suggested. Recordings may provide some benefits in terms of reflecting our impressions of the interaction and focusing more readily on the interview process and questions. This benefit is mentioned by Longhurst (2009). On the other hand Bhattacherjee (2012) says that the interviewer should also take notes even during the interview. This approach was embraced as it helped us to remember key points from the interviews. Audio recordings were used for Skype-based interviews and face-to-face interviews. Naturally, each interviewee was asked for permission to record for transcription purposes.

Qualitative researches tend to be conducted the natural settings (Creswell, 2007). In this case phone, face-to-face and audio conferencing were the alternatives that interviewee's could choose between. Face-to-face was preferred interviews in terms of having a more interactive conversation. Four of our respondents agreed to meet in person. One interview was conducted with audio conferencing, which surprisingly turned out to be three interviewees volunteering to participate. Telephone interviews are the best choices if the conditions are not suitable to have a face-to-face interview (Creswell, 2007). The advantage of this approach was obtaining a clear record without any noises in the background, since our only disadvantage conducting personal interviews was the selected places. The required time for transcribing the records is very much related with the quality of audio records (Kvale & Brinkmann, 2009). Although we did experience some difficulties dealing with noise it only delayed the transcribing process slightly.

One interview was conducted by mediated audio conferencing. The interview included three participants similar to a mini-focus group excluding the authors of this thesis (Onwuegbuzie, Dickinson, Leech & Zoran, 2009). Having a group interview allowed us to obtain three different views at the same time but not to find a solution or concurrence (Kvale & Brinkmann, 2009).

4.2.3 Selection of interviewees

Interviewees are often chosen purposively in qualitative research. Hence it is crucial to choose amongst identified representatives (Green & Thorogood, 2004). A three-stage process was devised to lure the most experienced interviewee's into participation. Therefore experts with a noted experience from CQRS were contacted. They were identified through short focus group with senior developers. Edument, a company facilitating training courses in CQRS, were also contacted as they have previously aided other authors in the subject (Niltoft & Pochill, 2013). In addition, past colleges with experience from CQRS were contacted. The contact was initiated by a personal address in an e-mail. Fortunately, the requests were well received and responses entailed participation in various forms; interview, technical supervision or aid. Between each interview the procured data was evaluated. Once enough material was gathered to describe CQRS requests for additional interviews were halted and the remaining interviews cancelled. This approach is supported by Kvale and Brinkman (2009) as they suggest the proper amount of interviews is determined by the data acquired. In total, five interviews were conducted with seven participants from across the different stages (see figure 4.1). It should be noted that the border between experts, educators and practitioners is blurry one at best. All of the participants could arguably be considered both educators and experts in their profession. All of the interviewees had previous experience from CQRS as this was considered qualifying for participation.

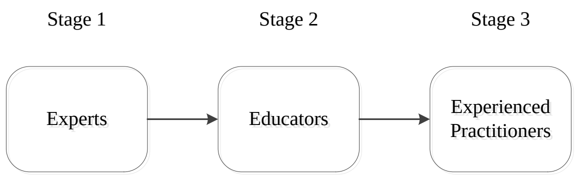

Figure 4.1 The three selection stages; experts, educators and practitioners

4.3 Data Analysis

The inductive nature of this inquiry mixed with exploratory approach and qualitative methods emphasizes rigorous familiarity with coding practices. To analyze the material the coding practices explained by Broom (2005) was utilized. It consisting of five stages; "as you go", "open coding", "axial coding", "selective coding" and "go back to theory part" (Broom, 2005, p. 71). Analyzing qualitative data is quite complex, since it requires a sophisticated knowledge and a good view in order to expose the obvious subjects that is being investigated, therefore the analysis was done in parallel to the data collection (Broom, 2005). By doing so the analysis was eased. Broom (2005) clarifies the approach help to develop initial themes and cope with complexity of the data gathered.

In the "open coding" stage generally each interview was reviewed at least two times by the authors of this thesis and taken notes on the transcription papers initially data was broken down according to the identified themes; opportunities and challenges (Broom, 2005). Moreover data was handled in order to expose the key ideas and concepts that are related to the topic which is the main aim of open coding process (Bhattacherjee, 2012).

The process of qualitative analysis starts with a question by question analyzing approach in order to see distribution of answers then researcher is ready to select some parts of interviewees' discourse (Talja, 1999). This required to group similar statements provided by respondents as a usual way of doing it for the purpose of describe and diversify statements about a specific topic (Talja, 1999). Additionally, similar statements were also grouped into according to the two identified themes. However, those themes were only utilized at this stage as an initial, broad classification.

Once this was done for all transcriptions, they were merged and sorted by the second theme that we assumed as a broader categorization comes from design pattern concept. Thus, self-categorization process was eased. While we were developing our own categorization list according to the meanings of statements, long quotes were summarized in few words. The brief statements were restated in order to make them as simple as possible by querying these meaning units for the specific purposes of the study. Eventually, final themes were bunched together into a descriptive statement. This is called "meaning condensation" that allows to deal with the data which is about ordinary language statements, without transforming data to the quantitative statements (Kvale & Brinkmann, 2009).

After the initial coding process, the themes were cross-referenced between transcriptions. This process is called "axial coding" (Broom, 2005, p. 71). Additionally, we went back to our notes and seek for a possible emerging inconsistency which is called "constant comparison" (Broom, 2005). Forward-backward movements helped us a lot in identification of common themes and give a chance to check each interview if some concepts exist in relation to the defined themes (Casterle, Gastmans, Bryon & Denier, 2012). This process can be conducted in the same time with open coding (Bhattacherjee, 2012). Therefore, this approach was important for refining the descriptive statements.

"Selecting coding" is about refining the data and considering each determined consistencies, categories and constructions (Broom, 2005). In this stage, we used our previous themes; opportunities and challenges (Bhattacherjee, 2012).

The last stage involved going back to the theory chapter and looking for ways of interpreting the data through the theoretical concepts (Broom, 2005). Generally, during the open coding phase, the comparisons between theory and data were made in terms of capturing matches and sometimes taking notes for the analysis part. This helped us a lot for developing the structure of our analysis and discussion sections earlier in the minds.

4.4 Research quality

Qualitative research isn't based on statistical work and formulation of them. Instead of quantification, it's preferably studied in a natural settings such as interviews, observation and case studies (Bashir, Afzal & Azeem, 2008). The quality of qualitative research can be elaborated in terms of validity and reliability. For many years these concepts were developed within quantitative approaches and later on adopted by qualitative researchers (Seale, 1999a). However, some scholars claim that validity in qualitative research is not entirely convenient to use in quantitative research. Considering validity was defined with words like plausible, credible and trustworthy (Bashir, Afzal & Azeem, 2008). Additionally, Lincoln and Guba proposed an alternative method in 1985 which includes four criterions such as credibility, transferability, dependability and confirmability that all correspond to one criteria in the old method (Qualitative Guideline Project, 2006).

Since we had conducted interviews including one group interview. The material collected was quite rich in text and several issues have to be considered. In order to achieve the quality of the study in a more effective way it was decided to consider the criterions of the two methods.

4.4.1 Reliability and dependability

Reliability refers to testing the study in quantitative approach, while testing means a way of information elicitation in qualitative approach and it is also accepted as quality of the research (Bashir, Afzal & Azeem, 2008). According to Stenbacka (2001, p. 551) "generating understanding" is the goal of qualitative studies when reliability is applied which is very valuable for our research since we focused on interviews (Golafshani, 2003). Choosing the interviewees through the true experts and compatibility of the interview questions were another issue considered in that respect. We tried to have respondents that they are not too irrelevant from each other in terms of their experiences. For generating understanding this was a key issue from the authors' point of view this thesis.

In addition having both individual and group interviews with the practitioners even if they were not with the same people, we tried to handle those two as "overlapping" methods for the purpose of ensuring dependability (Shenton, 2004). Definitely, it helped a lot more than conducting individual interviews; a general overview was gained after this interview which was also anticipated to evaluate some statements during the interview. In fact, capturing the similar statements was much easier than comparing the other individual interviews. That was quite understandable since there was an atmosphere like the conversation seemed not set up.

4.4.2 Validity and credibility

In qualitative research, validity means to investigate the accuracy of the findings and make certain if they are supported by evidence or not (Guion, Diehl & McDonald, 2002). We employed investigator triangulation for the interviews and the group interview in the analysis process. The findings were examined whether all of our investigators reach to the same conclusion or not, if so our confidence would be increased (Guion, Diehl & McDonald, 2002). In this process we utilized from the knowledge that we gained recently while we were examining the topic. Since credibility requires describing or understanding the phenomena from the participant's point of view (Qualitative Guideline Project, 2006). Unfortunately the eager attempts for using *member validation* where met with mild interest. Only one interviewee agreed to take a look at the transcripts and another one was interested in the quotes used. Although we were able to establish *construct validity* as mentioned by Yin (2003), as the research framework was derived from a comparison of descriptive templates used to describe recurring designs or patterns. It was imperative that it made sense to the interviewees and those relevant topics were dealt with. A separate part of the interview guide was used to establish the suitability of the questions asked. Specifically, it was asked if CQRS can be viewed as a recurring design or pattern. Additionally, the relevance of the questions asked was evaluated specifying necessary additions or omissions. Overall there only a few adaptions of the interview questions as were well-received and relevant according to the interviewees'. Although, some interviewees did point out that CQRS is a concept that may be even more profound than a pattern. It could also be regarded as an architectural style or principle.

4.4.3 Ethical issues

Anonymity and confidentiality are both crucial issues with regard to semi-structured interviews. Interviewees should be guaranteed proper security of the collected data (Longhurst, 2009). Thus, initially before the interviews by adding relevant permission questions and some information about the study into the interview guide we yielded awareness of the participants. The ethical guidelines concepts of Kvale and Brinkmann (2009) were utilized in this process. Firstly, the purpose of the study was explained and some information about the risks and benefits of participating to the study was provided. This is called informed consent by Kvale and Brinkmann (2009). After that, confidentiality was ensured by asking their permission for recording the conversation and making sure that if they are willing to share identifiable information. We asked if the participants have questions related with any issues regarding to the study in the beginning and in the end of the interviews. We believe that this is about responsibility of the researcher in terms of possible consequences and reducing the risks. The last concept of the ethical guideline is the role of researcher that we considered it by taking on a great effort for submitting the most accurate information from the results (Kvale & Brinkmann, 2009).

Based on the feedback received the recordings and transcripts would not be disclosed along with the thesis, instead they were made available as a separate document during peer-review. Analyses and citations were used with the permission of respondents. Personal names, company names and other identifiers and were obfuscated or omitted.

5 Findings

In this chapter we present our findings according to the categories that we have described in research framework chapter; opportunities and challenges. Themes were developed during the coding stage. These were alignment, simultaneous development, flexibility, modularization and complexity. The findings of this study are introduced with quotes and extended with discussions, rather than only describing without interpretation. This aids interpretation of the finding and makes the passages more fluent. The codes identify the interview and distinguish different interviewees (e.g. I2 or I5:2).

5.1 Opportunities

5.1.1 Alignment

CQRS is the segregation of read and write responsibilities within a component in an application. A component is usually a feature, module, or bounded contexts as termed by Evans (2004). The architectural result is a component that divides read and write logic into different parts.

> "In the end its two models one for read one for writes, when it comes to responsibilities and usually that ends up with a bunch of models for reads and the bunch of models for writes. You need them to align to use cases for your writes and for your reads. In one way, it's like taking the transactional model which we need to store data and then splitting it up to more fine-grained control for reads and writes and how we use them." (I5:1)

Similarly, two other interviewees' state;

> "Firstly the pattern offers an optimized model … for writes and an optimized model for reads which means that you can shape them both to be as good as possible for those two things." (I2)

> "CQRS is a simple principle where you maintain multiple models, usually one model for reads and one model for write. And this is what denormalization is." (I5:2)

The reason for the separation is the ability to align the read and write models to the use cases specified in the business requirements. Write models effectively maps to use cases that change data. Read models on the other hand maps to use cases that details user interfaces. For each use case to view some information there is a read model that corresponds to that view.

> "The thing about the read is that you usually want to align it with use case that you're reading on and that goes same for the writes … [When] you think about the models in CQRS, they should be mapped to some use cases, how they are performed, so they are aligned to the product I would say." (I5:1)

Explicitly interpreting the use cases in the solution usually means adding more source code to the application which naturally is something that has to be maintained and reduces source code reuse. However it is argued that the additions will be more aligned with the business requirements. This is something that often pays off in the long run. It is believed that it often reduces complexity down the road though it might be quite a leap to take the first step.

> "It may give you little less reuse when it comes to reusing the services better, on the other hand it will be more aligned to the product." (I5:1)

> "There will be a lot of different components for something that might have just been a single subsystem or a single component before … I claim that that the basic complexity of introducing [CQRS] is a win in the long run. It absorbs other types of complexity that will come later, infallibly in the development of an application. The first step can be hard to take." (I3)

In continuation, this alignment facilitates communication between development teams and business teams as they have a unified model that can be used by both teams. Having a common model that specifies commands and queries and is seen as a key factor that improves understanding and ease communication between stakeholders. Additionally a common model may guide system design as it captures user intent at an early stage.

> "It was much easier … for the technical teams and business teams to understand each other, because they had a model that could be understood by both of them." (I5:3)

> "[...] it is a very good idea to start designing the system in this way and [list] commands and queries in order to view what the system would be like. Because it establishes such a good overview and understanding of the system." (I3)

CQRS is suggested to facilitate better overview of system and improve communication between stakeholders in the development process. However it makes components less reusable. These two things have to be leveraged against each other. Stakeholders need to consider whether to emphasize communication and alignment of the IS to the use cases, or to favor code reuse.

5.1.2 Simultaneous development

Another theme is that CQRS enables multiple teams to work in parallel. Read and write models are different kinds of development tasks and can consequently be delegated differently. This opportunity was previously noted by Niltoft & Pochill (2012) as a benefit of CQRS. Those findings are collaborated in this inquiry.

> "When I was working at the big [company] where one part of the system, one team was actually working hard to maintain the transactional model and other teams were working on maintaining specific read models." (I5:3)

The write model is generally perceived as an integral part of the business domain that requires close collaboration with business teams and domain experts. Greater understanding of business processes and models are keys. Read models are simpler to maintain and usually appropriate for outsourcing.

> "Also you could put your senior developers and your business experts in the command side because that's where the business logic is. The read side you can outsource because the read side is usually not complicated to fix. It's simple." (I1)

Another interviewee agrees that read models are simpler to develop and are consequently more suitable for outsourcing;

> "From a team perspective there are the benefits. You can divide the responsibility … so developers that understands the business [work on write] while read only needs … a description of how to shape the data ... In theory this can be outsourced to someone who doesn't understand the business." (I2)

The possibility of dividing development work between different responsibilities was mentioned by three of the interviewees, although only one had explicit experience from do so. It was indicated that the amount of complexity surrounding write was substantially more than could be argued for reads. Write models are closely related to the core business domain and consequently should be developed by experienced developers and domain experts. Reads on the other hand could both be developed and maintained by outsourced teams.

5.1.3 Flexibility

A common concern among practitioners in this study is uncritical reliance on databases, especially RDBMSs. When relational databases are used it is common that relational consistency is over-emphasized. The end result may be a monolithic data model that improperly ensures relational consistency when relational boundaries should exists. Thus independence from specific database solutions is something that is greatly desired as it offers greater flexibility without inhibiting design decisions or locking the design.

> "People very dependent of the database. They have been influenced by the database to the extent where the database is calling the shots. Sometimes the damage shows in the source code, sometimes it shows in the user interface. Some combination of not seeing alternative ways to build systems. You have learnt to build databases and to make them normalized, creating foreign keys in a certain way and so on. Then there is no option pulls out when refactoring is due. You can only refactor all the tiers except for the database. This is where you're stuck. This is where you designed yourself into a corner." (I3)

Interestingly, a viable alternative to CQRS is to use a NoSQL-database. Mentions include RavenDB, MongoDB and CouchDB. Many NoSQL-databases sports a document store and multiple denormalized views on that document. This is not entirely dissimilar to having a separated write and read model in the database.

An interviewee explains;

> "[T]here are other ways almost CQRS for instance if you use RavenDB, for instance, which has data projections built-in. Then you almost get the best." (I2)

Another continues;

> "If we look at denormalization as a concept again. Most of the NoSQL stores like MongoDB, CouchDB. All of those take denormalization into consideration and your transactional model is the document and then you run map-reduce in order update your denormalized views. And that again is solved in the database at that point." (I5:1)

Such databases aren't equivalent to CQRS but may be simpler alternative. However coupling an application with a specific database-technology might inhibit a database switch later on. Relying on databases to do the denormalization is seen as unnecessary and limits the ability to design the application freely. An application that relies on built-in denormalization effectively couples the application with a specific database solution.

> "But when CQRS was defined it was more from the code point of view because we know that pushing all the responsibilities down to a database it's not very good. Because of the flexibility becomes quite bothering to be honest." (I5:1)

CQRS offers a solution similar to some NoSQL-databases but with added flexibility. In essence any kind of database could be used in collaboration with CQRS.

5.2 Challenges

5.2.1 Modularization

An issue that had been noticed in the past is ambitious overuse of CQRS. The concept is only appropriate in some parts of a system. These parts may be components, features, processes, modules or bounded contexts. Bounded context is a term coined by Evans (2004) and refers to an organization of parts within a system that usually collaborates with organization of business and development teams. But regardless of how modularization is accomplished CQRS should mainly be evaluated in relation to such a part.

One interviewee states;

> "CQRS is not a way to decompose your application [from the top-level] and shall never be. Because then you're really shooting yourself in the foot. And it will take time for you to recognize that but in the end it will be more pain than to have SQL than in the back to do that … If you're having CQRS on the top-level pattern and that might be a no-no in most cases, I would say." (I5:1)

Another interviewee continues;

> "It surely isn't something one should apply to the entire architecture uncritically. You do it in certain subsystems, often with certain properties. They are not pure CRUD-systems ... [I]t almost always is about a subsystem. You can talk about it as a bounded

> context or you could talk about it as services or something else. It considerably more common that you take such a part of the whole architecture and change." (I3)

This was very apparent in the results gathered and was mentioned by all interviewees to some extent. In fact one of the interviewees exemplified this by sketching out how such a modularization could look on a notepad. The sketch was modelled in the figure is shown below.

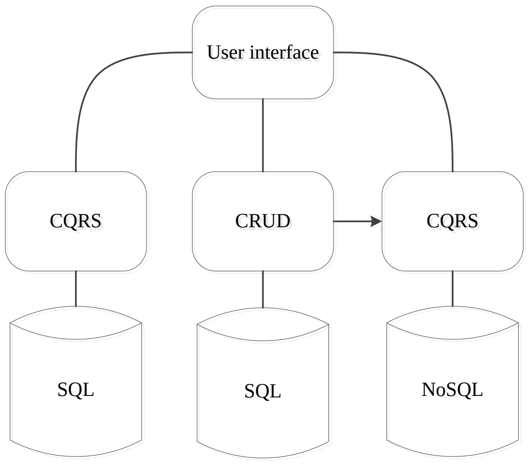

Figure 5.1 An example of an application with CQRS and CRUD deployed side-by-side as sketched by I1. Each part has a separate database and communication between is done in source code rather being distributed through the database.

The figure shows CQRS as one of many concepts in modular architecture. In this example each component relies on a separate database and even combines different types of databases. The complex relation between patterns and architecture was mentioned by Gamma et al. (1995). Patterns and architectures should collaborate in a way that makes sense. Similarly CQRS has to fit in the parenting architecture. It is generally considered to a mistake is to apply CQRS as top-level architectural principle rather than within a component. Consequently a modular architecture is suggested to be a prerequisite for successful implementation of CQRS.

5.2.2 Complexity

According to our findings it is not uncommon that write and read models reside in different places e.g. different tables or even different databases. This affects response times and performance in various ways since denormalized read models can be created ahead of time as opposed when queried. Consequently writing will take longer if all read databases are updated consistently.

> "So for example if you do all your writes to all the models within a single transaction it can make your writes right fairly expensive." (I5:3)

This is where eventual consistency comes into play. As long as the write model is saved in consistent manner then read models may be updated eventually.

> "[T]hat's where modern databases or alternative approaches come in whether you actually want to do something deferred or in asynchronous fashion from the transactional model. Like the read model doesn't need to be updated within this transaction it can be within one second that's good enough but one second for a machine is an enormous amount of time and it makes sense to not have everything in one transaction." (I5:1)

An important notion, as pointed out by Erb & Kargl (2014), is that the read model doesn't have to extend to the database. Read models may also be created in the more traditional sense; when it's queried. Thus application designers may choose between creating read models ahead of time, ahead of time but eventually, or when queried. To guide this decision a means of analysis has been identified. A common analysis is to establish the ratio between reads and writes. It may turn out it's simply worth to update the read models by taxing the writes, even if this is done in a consistent manner.

A general opinion that was aired is that complexity increases drastically when other concepts are mixed with CQRS. Complexity seems to stem from eventual consistency or event sourcing or a combination thereof. When these concepts are collaborated with CQRS it brings along tremendous complexity, which is indicated in the data gathered.

One interviewee cautions against adoption of CQRS as it usually entails adoption of other concepts as well;

> "So that's why I think it's the biggest risk at least according to me but CQRS is that you will not use CQRS as this pure simple pattern, but what you will end up with is a bus some kind of replicated state model on your application-level. And in the end that complexity is quite big in my opinion." (I5:1)

Another interviewee clarifies that adding other concepts indeed adds complexity but would not be considered simply CQRS;

> "When you start to add other concepts, of course [complexity] gets worse and big but this would not be CQRS anymore, this would be entirely a different concept." (I5:2)

Yet another side with the other interviewees;

> "Event sourcing and other patterns can help to solve this issue, but complexity increases." (I4)

According to one respondent CQRS is inferred by event sourcing, but the reverse doesn't apply.

> "It's just event sourcing per say is unusable without CQRS pretty much. That's a kind of a reverse relationship. But people don't think about that much. It seems like it is bi-directional." (I5:2)

The complexity noted in this inquiry collaborate the findings in a previous case study (Niltoft & Pochill, 2013). The authors claim CQRS is a fairly simple concept, but event sourcing and DDD are complex subjects. This concept enjoys benefits in some scenarios and weaknesses in others. Needless to say CQRS is similar to design patterns as it may introduce unwanted complexity (Gamma et al., 1995). CQRS is similar to CQS though at a different level of abstraction. However, results indicate that a common mistake is to assume it is suitable as a top-level architectural design.

6 Discussion

This section elaborates CQRS in terms of opportunities and challenges. Simplicity has guided the presentation and analysis of the empirical findings. The findings are grounded on arguments implied by the data and related to the research questions through the themes mentioned in Findings. Arguments in each theme are examined in terms of how they relate to CQRS. Moreover, some suggestions are made in relation to the challenges found.

6.1 Research question 1

What is the seasoned practitioners' view on opportunities that arise from adoption of command query responsibility segregation?

This inquiry has found three opportunities that arise from using CQRS;

- *Alignment* between and IT artifacts and business requirements
- *Simultaneous development* that effectively allows multiple teams to work in parallel.
- *Flexibility* to adopt any kind of database technology.

Alignment refers to an IT artifact's relationship to a product's functional and non-functional requirements. This is achieved by mapping source code to use cases for both reads and writes. Since use cases are introduced in source code on a per-use case level, technical considerations can be altered for each use case depending on characteristics of the command or query. In this way CQRS offers fine-grained control of the design and execution of each use case. It does so by also reducing reusability as more components are added to the solution. These two things have to be leverage against each other. One-to-one relationship between use cases and read or write models has the effect that source code has to be adjusted when use cases are changed. The alignment also has 'softer' benefits. It plays a key role in creating a common understanding and a language that can be used by stakeholders especially; business teams, domain experts and development teams. This is a collaborative effect that indicate the close connection to *bounded contexts* and *ubiquitous language* as mentioned by Evans (2004).

Simultaneous development refers to the ability of teams to work in parallel. Like alignment this is also DDD aspect as mentioned by Betts et al. (2012). Evans (2004) suggests that technical teams should match the organization of business teams. This is something that should be reflected in IT artifact as well. It's basically the concept phrased as "notable boundaries" by Niltoft & Pochill (2013, p. 38). Similar to the case study by Niltoft & Pochill (2013) our findings indicate that CQRS allows for multiple development teams to work in parallel. The case study concluded that the business domain is divided to lesser part which makes it easier to delegate development work to different teams. Additionally, our results also indicate that there is a distinct difference between of commands and queries. Commands are 'guarded' as they contain the core business logic as well as maintain the transactional model. Reads on the other hand, are simple both from a maintenance and development perspective. A simple description of how to shape the data may suffice. The fact that read models are simple to develop was also hinted in a case study by Fitzgerald (2012). Thus, the read model is perceived as more appropriate for outsourcing.

A common worry was implications that occur when technical teams rely on database technology to make design decisions. Speaking from past experiences our interviewees' state that a particular database technology tends to govern the design of the code. This is seen as detrimental to the ISD process since it inhibits design decisions and locks the design to a particular database technology. CQRS comes from the notion that independence from a particular database is good. CQRS offers *flexibility* from specific databases and may use a relational database or NoSQL-database or both. Denormalized read models, when needed, are usually done in code anyway. As mentioned by Vajk el al. (2013) denormalization and normalization are concepts from different kinds of the databases. But CQRS isn't limited itself to choose between either normalization or denormalization due to decisions made on behalf of the application. CQRS may denormalize queries when queried, ahead of time or ahead of time but eventually consistently.

6.2 Research question 2

What is the seasoned practitioners' view on challenges that arise from adoption of command query responsibility segregation?

In spite that many difficulties were noted when applying CQRS we deal with two topics in this passage. There are two main challenges identified from the results in the thesis;

- *Modularization* of architectures is strongly advised, if not required, for adoption CQRS
- *Complexity* may become unmanageable if CQRS is incautiously extended with other concepts

A common challenge is the appropriate scope for CQRS according to the interviewees. If CQRS is adopted as a top-level-architecture it will often be an unfortunate mistake. Instead the concept can suitably fit into some subsystems. For instance, components, features, processes, modules and bounded contexts would be the parts of a system that are well-suited. It might be used in some parts of a system, and that is perfectly fine. It may also be combined with other patterns especially eventual consistency or event sourcing in some cases. Hence, to keep complexity limited to their subsystems a modularized structure is essential in order successfully implement CQRS.

Complexity is a second challenge. Interestingly, CQRS is regarded as a simple concept that reduces domain complexity. Yet interviewees stated that complexity is increased whenever concepts like eventual consistency and event sourcing are introduced. In agreement with conclusion by Niltoft & Pochill (2013) reads may be eventual consistency and does offer availability over consistency as CAP theorem suggests. However our findings also suggest that availability might be leveraged between reads and writes when denormalization is done (consistently) ahead of querying. This corresponds with Brewer (2012) claims that CAP theorem is more complicated model than simply choosing between tuples of consistency-availability and consistency-partition tolerance.

6.3 Conclusions

This investigation defines CQRS in terms of opportunities and challenges and suggests five things that could be to be considered. Although there might be many more aspects that also needs consideration these were the ones that were collaborated from the data gathered. By the nature of this inquiry the result presented here should be seen as indicative rather than generalizable or established. The opportunities are (i) alignment, (ii) simultaneous development and (iii) flexibility.

i. *Alignment* between IT-artifacts and business requirement helps to establish a common understanding that can be shared between business and development teams. It also makes it possible to explicitly map use cases to models used in source code and facilitates a ubiquitous language.
ii. *Simultaneous development* allows for multiple teams to work in parallel. As reads are 'simpler' they don't require comprehensive business know-how to maintain or develop.
iii. *Flexibility* to use any database solution is important. This limits the impact of particular databases on the application design.

Challenges indicated are (iv) modularization and (v) complexity;

iv. *Modularization* is an important concept, if not a prerequisite, for successful adoption of CQRS. It allows for CQRS to be adopted in some parts, but not other.
v. The potential introduction of unwanted *complexity* is a serious concern that may cause issues down the road. Because CQRS is regarded as an enabler for eventual consistent read models, it may cause a venture into that direction prematurely. Eventual consistency, as of today, still is in its infancy and effectively makes the application distributed. CQRS may also be an enabler for event sourcing; a concept that, similarly to eventual consistency, is new and inherently complex.

In summary, CQRS puts reemphasis on the collaboration between business teams, development teams where needed. It focuses on alignment between an IT-artifact and core business values. It provides more fine-grained control of the use cases and the technical considerations that needs to be met. According to the data collected it might be hard to develop a solution that aligns well to the use cases and non-functional requirements. By adopting CQRS these issue may become simpler, but may also introduce great risk if extended.

The findings indicate that the tools and techniques used to decompose complexities encountered in ISD sometimes are far too hard to elaborate by relying simply on a single model. Instead alternate perspectives on data and relational boundaries are emphasized. The relationships between inputs and outputs are more clearly specified. Untangling actions (commands) from information needs (queries) may convey correspondence between business processes and IT-artifacts. During the last decades preference towards RDBMS seems to have had tremendous impact on software, limiting critical thinking around the failures of such approaches. Moderns systems, especially those with complex user interaction, e.g. collaborate domains needs to decompose the models further to keep the IS aligned with the business domain, use cases and technical considerations. In such domains the focus seems to shift from having one model to multiple smaller models or "complex services" as phrased by Avison & Fitzgerald (2006).

6.4 Future research

6.4.1 Extract, transform, load

Several patterns from ETL and data warehousing were hinted at times. Perhaps patterns or theory found in those areas are applicable to CQRS. ETL's ability to eradicate errors, correct data, obtain trust and combine data sets through user tools (Xavier & Moreira, 2013), could add value to CQRS or event sourcing. The feasibility of having a user tool to generate transition between states has already been suggested (Fitzgerald, 2012).

6.4.2 Operational transforms

The ability to add time-dependent logic that resolves concurrency issues using event sourcing was suggested by several interviewees. This capability was commonly termed as "merging commands". Operational transforms was explicitly hinted as a concept that solves similar issues although in a different context, namely wiki's and collaborate document environments. This theme was discarded as it related more closely to event sourcing than CQRS.

6.4.3 Probabilistic bounded staleness

As previously claimed by Hakim (2012) probabilistic bounded staleness (PBS) may be a suitable means to measure staleness in eventually consistent CQRS-components. Additionally, PBS has been suggested as general way to measure eventual consistency distributed databases as well (Golab et al., 2014)

6.4.4 Occasionally connected systems

Close relation between CQRS and event sourcing, and occasionally connected systems was hinted by two interviewees. Such systems would likely have to balance consistency, availability, partition tolerance with great care. An interesting avenue as some IS are deployed in unreliable networks.

7 Appendices

7.1 Interview guide

The purpose of the study
We aim to describe CQRS as pattern in a master's thesis. It seems like a good place to start as it relates to many interesting topics that could be studied sometime in the future. We hope to provide some detail on the intent, motivation, alternatives and applicability of the pattern along with some examples.

General questions
1. Is it ok if the conversation is recorded for transcription purposes?
2. Can you tell us a little bit about yourself?
3. How did you become aware of CQRS?
4. Do you have any questions about the interview?

CQRS questions
5. What are the key issues or problems addressed by CQRS?
6. When do you think one should consider using CQRS?
7. When should one avoid CQRS?

8. What different objects or classes make out this pattern?
9. How do their responsibilities differ?

10. What are the benefits of using CQRS?
    a. From a developer's perspective.
    b. From a business' perspective
11. What are the disadvantages or difficulties of using to the pattern?

14. Can you suggest one or several scenarios when CQRS is good fit?
15. How does CQRS solve issues in this (these) scenario(s)?

16. Can you think of any concepts that are closely related to CQRS?
17. Do any of them need special consideration?

18. Can you think of any patterns commonly are used in collaboration with CQRS?
19. Do you know of any other patterns that solve the same issues or parts thereof?

20. Can you suggest a code sample or library that exemplifies CQRS in a succinct fashion?

21. Are you familiar with Command Query Separation suggested by Bertrand Meyer?
    c. Bertrand Meyer claimed that CQS provided some guidance on how to do distributed computing.
    d. Do you think the same applies to CQRS in some way?
22. CQRS is often mentioned in relation to scalability concerns. Why do you think that is?

Closing questions
23. We intend to frame CQRS as a design pattern? Do you believe this is an appropriate framing for this topic?
24. Do you think we covered the most important topics related to CQRS?
a. What should be removed?
b. What should be added?

Debrief
The thesis will be available in the public domain. This has the consequence some of the information you share with us may be publicly available, in one form or another. Any recordings and transcriptions made will not be included in the thesis. It may however be subjected to inspection by a third party. Yet we may use direct quotes to maintain expressiveness from our conversation.

- Is this ok with you?
- Do you wish to be anonymous or to have your name appear in the study?
- Would you like to review the transcription of the interview?
- We have no further questions. Is there anything else that you would like to ask or add?

References

Abadi, D.J. (2012). Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story. IEEE Computer Society, Available through: http://www.lub.lu.se/ [Accessed 13 April 2014]

Abdullin. (2010). Software Design, 23 March2010. DDDD, CQRS and Other Enterprise Development Buzz-words: Blog. Available Online http://abdullin.com/ [Accessed 14 August 2014]

Ampatzoglou, A., Stamelos, I. & Frantzeskou, G. (2012). A methodology to assess the impact of design patterns on software quality, *Information and Software Technology*, vol. 54, Is. 4, pp. 331-346, Available through: http://www.lub.lu.se/ [Accessed 21 May 2014]

Avgeriou, P. & Zdun, U. (2005). Architectural Patterns Revisited–A Pattern [pdf]. Available at: http://www.infosys.tuwien.ac.at/staff/zdun/publications/ArchPatterns.pdf [Accessed 11 April 2014]

Avison, D. & Fitzgerald, G. (2003). 'Where Now for Development Methodologies?', Communications Of The ACM, 46, 1, pp. 78-82, Business Source Complete, EBSCOhost, [Accessed 1 July 2014]

Avison, D. & Fitzgerald, G. (2006). Information systems development: methodologies, techniques and tools, 4th edn, Berkshire: McGraw Hill.

Avram, A. & Marinescu, F. (2006). Domain Driven Design Quickly, [e-book] InfoQ Enterprise Development Series, Available at: http://www.infoq.com/minibooks/domain-driven-design-quickly [Accessed 10 April 2014]

Bailis, P. & Ghodsi, A. (2013). Eventual Consistency Today: Limitations, Extensions, and Beyond, Communications of The ACM, Vol. 56, Iss. 5, pp. 55 – 63. Available through: EBSCOhost [Accessed 26 July 2014]

Barry, C., Conboy, K., Lang, M., Wojtkowski, G. & Wojtkowski, W. (2009). Information Systems Development. Springer London.

Bashir, M., Afzal, M.T. & Azeem, M. (2008). Reliability and Validity of Qualitative and Operational Research Paradigm, *Pakistan Journal of Statistics and Operation Research*, vol.4, no.1, pp.35-44, Available at: http://www.pjsor.com/index.php/pjsor/article/viewFile/59/38scientific [Accessed 13 May 2014]

Betts, D., Domínguez, J., Melnik, G., Simonazzi, F. & Subramanian, M. (2012). Exploring CQRS and Event Sourcing - A journey into high scalability, availability, and maintainability with Windows Azure, MSDN document [pdf] Available at: http://msdn.microsoft.com/en-us/library/jj554200.aspx [Accessed 11 February 2014]

Bhattacherjee, A. (2012). Social Science Research: Principles, Methods, and Practices [e-book] USF Tampa Bay Open Access Textbooks Collection, Available at: http://scholarcommons.usf.edu/oa_textbooks/3 [Accessed 13 May 2014]

Bogard. (2012). 22 August 2012. Busting some CQRS myths: Blog. Available online: http://lostechies.com/ [Accessed 27 July 2014]

Brewer, E. (2001). Lessons from giant-scale services. IEEE Internet Computing, 5, 4, p. 46-55, Scopus®, EBSCOhost, [Accessed 29 June 2014]

Brewer, E. (2012). CAP Twelve Years Later: How the "Rules" Have Changed. IEEE Computer Society, Available through: http://www.lub.lu.se/ [Accessed 11 April 2014]

Broom, A. (2005). Using qualitative interviews in CAM research: A guide to study design, data collection and data analysis, Complementary Therapies in Medicine, vol. 13, pp. 65 – 73, Available at: http://www.complementarytherapiesinmedicine.com/article/S0965-2299(05)00009-9/abstract [Accessed 13 May 2014]

Buschmann, F., Meunier, R., Rohnert H., Sommerland, P. & Stal M. (1996). Pattern-Oriented Software Architecture. A system of patterns. Chichester: Wiley.

Casterle, B.D., Gastmans, C., Bryon, E. & Denier, Y. (2012). QUAGOL: A guide for qualitative data analysis, International Journal of Nursing Studies, vol. 49, pp. 360 – 371, Available at: http://www.journalofnursingstudies.com/article/S0020-7489(11)00367-1/abstract [13 May 2014]

Creswell, J.W. (2007). Qualitative Inquiry & Research Design: Choosing Among Five Approaches, 2nd ed, Thousand Oaks: Sage Publications

Dahan, U. (2011). 22 April 2011. Udi Dahan – The Software Simplicit, When to avoid CQRS: Blog. Available online: http://www.udidahan.com/2011/04/22/when-to-avoid-cqrs/ [Accessed 10 March 2014]

Domínguez, J., Melnik G. (2012). A Journey into CQRS, [video online] Available at: http://channel9.msdn.com/Events/Patterns-Practices-Symposium-Online/Patterns-Practices-Symposium-Online-2012/A-Journey-into-CQRS [Accessed 10 March 2014]

Doody, O. & Noonan, M. (2013). Preparing and conducting interviews to collect data. *Nurse Researcher,* vol.20, no.5, pp. 28-32, Available at: http://rcnpublishing.com/doi/pdfplus/10.7748/nr2013.05.20.5.28.e327 [Accessed 13 May 2014]

Erb, B. & Kargl, F. (2014). Combining discrete event simulations and event sourcing. In *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques* (pp. 51-55). ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). [Accessed 29 July 2014]

Evans, E. (2004). Domain-Driven Design: Tackling Complexity in the Heart of Software, Boston: Addison - Wesley

Fitzgerald, S. (2012). State Machine Design, Persistence and Code Generation using a Visual Workbench, Event Sourcing and CQRS, MSc thesis, School of Computer Science and Informatics, University College Dublin, Available online: http://thesis-latex.googlecode.com/svn/trunk/cqrs-thesis.pdf [Accessed 4 June 2014]

Flick, U., Kardoff, E. & Steinke, I. (2004). A Companion to Qualitative Research, [e-book] Glasgow: Sage Publications. Available through: Google Books: books.google.com [Accessed 13 May 2014]

Fountoukis, S. & Chatzistavrou, D. (2012). Pattern oriented design of cluster running object medical information systems, AIP Conference Proceedings, vol. 1504, Is. 1, p. 1248, Available through: http://www.lub.lu.se/ [Accessed 21 May 2014]

Fowler, M. (2003). Patterns of Enterprise Application Architecture, Boston: Addison-Wesley.

Fowler, M. (2005). Martin Fowler, 12 December 2005. Event sourcing: Blog. Available at: http://martinfowler.com/eaaDev/EventSourcing.html [Accessed 4 August 2014]

Fowler, M. (2011). Martin Fowler, 14 July 2011. CQRS: Blog. Available at: http://martinfowler.com/bliki/CQRS.html [Accessed 26 May 2014]

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley.

Gilbert, S.S. & Lynch, N. N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant Web services. *SIGACT News*, *33*(2), 51-59. [Accessed 28 May 2014]

Gilbert, S. & Lynch, N.A. (2012). Perspectives on the CAP Theorem. *IEEE Computer Society*, vol. 45, no. 2, pp. 30 – 36. Available through: http://www.lub.lu.se/ [Accessed 13 April 2014]

Golab, W., Rahman, M., Auyoung, A., Keeton, K. & Xiaozhou, L. (2014). Eventually consistent: not what you were expecting? Communications of The ACM, vol. 57, Is. 3, pp. 38-44, Available through: http://www.lub.lu.se/ [Accessed 26 May 2014]

Golafshani, N. (2003). Understanding Reliability and Validity in Qualitative Research, *The Qualitative Report*, vol.8, no.4, pp.597-607, Available at: http://www.nova.edu/ssss/QR/QR8-4/golafshani.pdf [Accessed 13 May 2014]

Green, J. & Thorogood, N. (2004). Qualitative Methods for Health Research, [e-book] Wallington: Sage Publications. Available through: Google Books: books.google.com [Accessed 13 May 2014]

Green, S., (2010). Captain Codeman .NET Developer, March 31, 2010. Homongous! MongoDB, NoSQL and CQRS. Blog: Available online:

http://captaincodeman.com/2010/03/31/homongous-mongodb-nosql-cqrs/ [Accessed 1 June 2014]

Guion, L.A., Diehl, D.C. & McDonald, D. (2002). Triangulation: Establishing the Validity of Qualitative Studies, [pdf] Available at: http://edis.ifas.ufl.edu/fy394 [Accessed 13 May 2014]

Gustavsson, S. & Andler, S. F. (2002). Self-stabilization and eventual consistency in replicated real-time databases. In Proceedings of the first workshop on Self-healing systems, pp. 105-107, Available through: http://www.lub.lu.se/ [Accessed 4 June 2014]

Hakim, K. (2012). Correctness for CQRS Systems: Elicitation and validation, MSc thesis, School of Computer Science and Communication, KTH, Available online: http://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2012/rapporter12/hakim_kamil_12072.pdf [Accessed 4 June 2014]

Horton, J., Macve, R. & Struyven, G. (2004). Qualitative Research: Experiences in Using Semi-Structured Interviews in C. Humphrey & B. Lee (eds), The Real Life Guide to Accounting Research: A Behind-the-Scenes View of Using Qualitative Research Methods, [e-book] pp. 339-357, The Netherlands: Elseiver. Available through: Google Books: books.google.com [Accessed 13 May 2014]

Jain, P. & Kircher, M. (2004) Pattern-Oriented Software Architecture: Patterns for Resource Management; Volume 3. Munich: Wiley.

Kabbedijk, J., Jansen, S. & Brinkkemper, S. (2013). A Case Study of the Variability Consequences of the CQRS Pattern in Online Business Software, Proceedings of the 17th European Conference on Pattern Languages of Programs, Article no. 2, Available through: ACM Library [Accessed 14 August 2014]

Khachana, R., James, A. & Iqbal, R. (2011). Relaxation of ACID properties in AuTrA, The adaptive user-defined transaction relaxing approach, Future Generation Computer Systems, 27, 1, pp. 58-66, Inspec, EBSCOhost, [Accessed 29 June 2014].

Kvale S. & Brinkmann, S. (2009). Interviews: Learning the Craft of Qualitative Research Interviewing, 2nd edn, USA: Sage Publications.

Kvale, S. (2006). Dominance through Interviews and Dialogues, *Qualitative Inquiry*, vol. 13, no.3, pp. 480 – 500, Available through: http://www.lub.lu.se/ [Accessed 22 January 2014]

Lloyd, W., Freedman M., Kaminsky M. & Anderson, D. (2014). Don't Settle for Eventual Consistency, *Communications of The ACM*, vol. 57, Is. 5, pp. 61-68, Available through: http://www.lub.lu.se/ [Accessed 26 May 2014]

Longhurst, R. (2009). Interviews: In-Depth, Semi-Structured in R. Kitchin & N. Thrift (eds), *International Encyclopedia of Human Geography*, pp.580-584, Available through: Science Direct, [Accessed 10 May 2014]

Malarvannan, M. & Ramaswamy, S. (2010). Rapid Scalability of Complex and Dynamic Web-Based, *2010 5th International Conference on System of Systems Engineering (SoSE)*, pp. 1 – 6, Available through: IEEEXplore Digital Library [Accessed 06 March 2014]

Malcangi, M. A. (2013). Application in medical system, Data Fusion from Sensors for Context-aware, Healthcare-management Systems. Available online: http://air.unimi.it/handle/2434/228844 [Accessed 14 April 2014]

Martin R. C. M. (2006). Agile Principles, Patterns, and Practices in C Sharp, Prentice Hall, 2006.

Meyer, B. (1988). Object-oriented software construction, New Jersey: Prentice Hall

Meyer, B. (1997). Object-oriented software construction, 2nd edn, New Jersey: Prentice Hall

Monroe, R.T., Kompanek, A., Melton, R. & Garlan, D.B. (1996). Architectural Styles, Design Patterns, and Objects, *Tepper School of Business*, Available at: http://repository.cmu.edu/tepper/318 [Accessed 11 April 2014]

MSDN Library (2014). Command and Query Responsibility Segregation (CQRS) Pattern. Available online: http://msdn.microsoft.com/en-us/library/dn568103.aspx [Accessed 03 July 2014]

Myers, M.D. & Newman, M. (2007). The qualitative interview in IS research: Examining the craft, *Information and Organization*, vol.17, pp. 2 – 26, Available through: Science Direct [Accessed 13 May 2014]

Nijhof, M. (2014). CQRS, [e-book] Lean Publishing, Available at: https://leanpub.com/cqrs [Accessed 12 May 2014]

Niltoft, P. & Pochill, P. (2013). Evaluating Command Query Responsibility Segregation. Lund: Department of Computer Science, Faculty of Engineering, LTH, Lund University

Oliver. (2010). 22 February 2010. Event Sourcing and CAP Requirements: Blog. Available Online: http://blog.jonathanoliver.com/event-sourcing-and-cap-requirements/ [Accessed 14 August 2014]

Onwuegbuzie, J.A., Dickinson, W.A., Leech, N.L. & Zoran, A.G., (2009). A Qualitative Framework for Collecting and Analyzing Data in Focus Group Research, *International Journal of Qualitative Methods*, vol.8, no.3, pp.1-21, Available through: http://www.lub.lu.se/ [Accessed 13 May 2014]

Parse, R.R. (2001). Qualitative Inquiry: The Path of Sciencing, [e-book] Sudbury, MA: Jones and Barlett Publishers, Available at: Google Books: books.google.com [Accessed 13 May 2014]

Qualitative Guideline Project (2006). Lincoln and Guba's Evaluative Criteria, Available online: http://www.qualres.org/HomeLinc-3684.html [Accessed 13 May 2014]

Rajkovic, P., Jankovic, D. & Milenkovic, A. (2013). A software model of mobile notification system for medication misuse prevention. Department of Computer Science, University of Niš, Faculty of Electronic Engineering, Republic of Serbia. In e-Health Networking, Applications & Services (Healthcom), 2013 IEEE 15th International Conference on (pp. 569-574). IEEE. Available through: http://www.lub.lu.se/ [Accessed 4 June 2014]

Rajković, P., Janković, D. & Milenković, A. (2013). Using CQRS Pattern for Improving Performances in Medical Information Systems, *The 6th Balkan Conference in Informatics (BCI 2013)*, pp. 86 – 91, Available at: http://ceur-ws.org/Vol-1036/p86-Rajkovic.pdf [Accessed 4 June 2014]

Recker, J. (2013). Scientific Research in Information Systems – A Beginner's Guide, [e-book] London: Springer. Available at: http://link.springer.com/book/10.1007/978-3-642-30048-6 [Accessed 13 May 2014]

Robert. (2013). 21 March 2013. Introduction to CQRS: Blog. Available Online: http://www.codeproject.com/ [Accessed 27 July 2014]

Sandelowski, M. (2000). Focus on Research Methods WhateverHappened to Qualitative Description? *John Wiley & Sons, Inc. Research in Nursing & Health,* 23, pp. 334-340, Available at: http://www.wou.edu/~mcgladm/Quantitative%20Methods/optional%20stuff/qualitative%20description.pdf [Accessed 13 May 2014]

Seale, C. (1999a). The Quality of Qualitative Research [e-book] London: Sage Publications, Available at: Google Books: books.google.com [Accessed 13 May 2014]

Seale, C. (1999b). Quality in Qualitative Research, *Qualitative Inquiry,* vol.5, no.4, pp.465-478, Available at: http://qix.sagepub.com/ [Accessed 13 May 2014]

Shenton, A.K., (2004). Strategies for ensuring trustworthiness in qualitative research projects, Division of Information and Communication Studies, vol. 22, pp. 63 – 75, Available at: https://xa.yimg.com/kq/groups/73868647/750861395/name/Trustworthypaper.pdf [28 May 2014]

Simon, H.A. (1996). The Sciences of the Artificial. Cambridge: MIT Press

Stenbacka, C. (2001). Qualitative research requires quality concepts of its own, *Management Decision,* vol. 39, Iss: 7, pp. 551 – 556, Available through: http://www.deepdyve.com/lp/emerald-publishing/qualitative-research-requires-quality-concepts-of-its-own-1i6VJIw15G [Accessed 4 June 2014]

Talja, S. (1999). Analyzing Qualitative Interview Data: The Discourse Analytic Method, *Library & Information Science Research,* vol. 21, no. 4, pp. 459 – 479, Available at: https://www.zotero.org/ion/items/itemKey/ZZT5IC2P , [Accessed 20 May 2014]

Tracey, M., Hutchinson, A. & Grzebyk, T. (2014). Instructional designers as reflective practitioners: developing professional identity through reflection, *Educational Technology Research and Development*, pp. 1-20, Available through: Scopus®, EBSCOhost [Accessed 23 May 2014]

Truyers, K. (2013). 5 December 2013. Introducction to Domain Driven Design, CQRS and Event Sourcing: Blog. Available Online: http://www.kenneth-truyers.net/category/blog/ [Accessed 14 August 2014]

Vajk, T, Feher, P, Fekete, K. & Charaf, H 2013, 'Denormalizing data into schema-free databases', Available through: EBSCOhost, [Accessed 17 May 2014]

Wang, X., Sun, H., Deng, T. & Huai, J. (2013). Consistency or latency? A quantitative analysis of replication systems based on replicated state machines, Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference, pp. 1-12. Available through: IEEE Ludwig Lub Search [Accessed 26 July 2014]

Wendorff, P. (2001). Assessment of design patterns during software reengineering: lessons learned from a large commercial project. *Proceedings Fifth European Conference on Software Maintenance & Reengineering*, pp. 77 – 84, Available through: http://www.lub.lu.se/ [Accessed 4 May 2014]

Vesilecas, O., Caplinskas, A., Wojtkowski W., Wojtkowski W.G., Zupancic, J. & Wrycza S. (2004). Information Systems Development, Advances in Theory, Practice and Education [e-book] Springer: Lithuania. Available through: http://www.springer.com/?SGWID=5-102-0-0-0#mainnav-headerLogo [Accessed 14 August 2014]

Vogels, W. (2009). Eventually consistent. Communications of the ACM, 52(1), 40-44. Available through: ACM [Accessed 14 July 2014]

Xavier, C. & Moreira, F. (2013). Agile ETL, Procedia Technology, Vol. 9, pp. 381 – 387. Available through: Science Direct [Accessed 13 August 2014]

Yin, R. K., 2003, Case study research: design and methods, Thousand Oaks: Sage

Young G., (2010). February 16, 2010. Code Better. CQRS, Task Based UIs, Event Sourcing agh! Blog: Available online: http://codebetter.com/gregyoung/2010/02/16/cqrs-task-based-uis-event-sourcing-agh/, [Accessed 12 March 2014]

Young, G. (2010). CQRS Files. CQRS Documents by Greg Young [pdf] Available at: http://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf [Accessed 27 July 2014]

Zhang, C. & Budgen, D. (2012). What Do We Know About the Effectiveness of Software Design Patterns? *Software Engineering, IEEE Transactions on*, vol. 38, Is. 5, pp. 1213-1231, Available through: IEEE Computer Society [Accessed 4 March 2014]