Graduate Business School

Masters Thesis 2000:42

Identifying Software Engineers' Competence at Iquity Systems
-Using a Phenomenographic Approach

Mark Castellino & David Hellström

Graduate Business School
School of Economics and Commercial Law
Göteborg University
ISSN 1403-851X
Printed by Novum Grafiska

Abstract

Technological change, intense global competition and the increasing emergence of knowledge-intensive companies have put new demands on management in organisations, which have led to a shift from valuing physical assets to more focusing on humans and their competence.

But, in this highly debatable field of competence, academics are arguing about how to define what it actually is while organisations, which recognise the importance of attracting and developing competent employees as the key for future success, are implementing a system or strategy that gives an illusion of competence within the field of management.

In this research, we used a phenomenographic approach to identify what competence is for Software Engineers at Iquity Systems Inc. This method allowed us to attain the Software Engineers' descriptions and experiences of working as a Software Engineer.

Our major finding could be seen as quite revolutionary in the field of competence, as we found that competence could be seen in another way than the traditional rationalistic view, looking at attributes necessary for accomplishing work. Instead, we argue that an individual's understanding/conception of work organises and delimits his way of accomplishing work.

Key words: Competence, phenomenography, interpretative, rationalistic, social construction, Software Engineers, Iquity Systems Inc.

Acknowledgement

We have, for this thesis, been given the opportunity to look into something that could be seen as somewhat unique, hence as our tutor expressed it – "on virgin soil". In our quest to investigate what competence is for Software engineers, we have several to give our deepest thanks and gratitude to.

Firstly, we wanted to thank the people at Iquity Systems Inc. for being so helpful and supportive in our thesis and we would give our special thanks to the Human Resource Manager Josefina Rosengren, as well as the Chief Technical Officer Niclas Ohlsson for giving us the opportunity to do our research at the company. We also would like to give our gratitude to all Software engineers that so enthusiastically participated in the interviews and discussions.

Also, we want to take the moment to thank Lena Wästfelt and Kristina Elliot at ConcoursCepro for giving us feedback and participating in long but interesting discussions concerning the topic. We would also show our appreciation to Peter Dickson at KnowIT for the continuous feedback in the analysis phase of our thesis.

Lastly, we want to give our tutors a very special thanks, Axel Targama and Torbjörn Stjernberg at Gothenburg School of Economics and Commercial Law, for giving us very valuable advice and insights in this fairly debatable issue, as shown by our tutors as well. Also, we should not forget to thank our class mates, at the Master of International Management (MIM) program, that have on a continuous basis given us both support and feedback on our topic at hand.

Mark Castellino and David Hellstrom
Gothenburg/Stockholm, Sweden
December 2000

TABLE OF CONTENTS

1 Introduction

"...To know when you know and to know when you don't know is knowledge..." - Confucius-

1.1 Background

Back in the 1950s, Drucker predicted the emergence of the so-called knowledge worker. Up until now, his vision has been realised as work continues to become more and more complex, non-standardised and dependent on the combination of numerous sources of knowledge (Wenger, 1998). Technology in for example communications and microelectronics is developing rapidly, requiring organisations to continually develop, attract and renew competence.

The changes taking place in today's business environment, such as technological change, intense global competition and an increasing emergence of knowledge-intensive companies have put new demands on the management of organisations. Blackler (1995) argues that in recent years, organisations are more dependent on managing knowledge and competence as opposed to bureaucratic control of resources.

Organisations' strengths are no longer only valued by their physical assets with standardised and routine work, but what is considered the key component to be successful is rather focusing on core competencies (Prahalad and Hamel, 1990) on the company level and human competence on the individual level (Sandberg, 1994).

1.2 Changing demands

Nowadays, terms like attracting, developing and retaining competent employees are frequently mentioned as today's main problems that most organisations are faced with in order to achieve or even remain in today's competitive marketplace. A structural change is taking place in Western economies as a result of a movement towards more service and knowledge-based industries, increasing the importance of competence development (Sandberg, 1994).

Since most industries are becoming more and more knowledge-intensive, jobs are becoming increasingly complex resulting in tasks characterised by uncertainty and improvisation rather than standardised tasks. Consequently, tasks require both different and more advanced levels of competence and an ongoing need for competence development.

In management, the concept of competence has gained great attention since it focuses on the relation between employee and work rather than on knowledge itself. In the quest of attracting and developing competent employees, it seems reasonable to look at what is the knowledge and skills needed in accomplishing work, which we refer to as competence.

At the same time as the concept of competence is gaining great attention as a result of organisations' quest for a competitive advantage, many companies have run into difficulties in applying the concept of competence in organisations since for example it is not that easy to define and measure. As a result, it is problematic for organisations to draw benefits from it. In addition, organisations have fundamental problems in developing human competence at work in a way that enables an organisation to remain viable (Sandberg 2000), still the majority of organisations today talk about the importance of the concept of competence. Instead of looking at competence in organisations, this has in many situations given the concept of competence development an undeserved use as a synonym for education (Sandberg & Targama, 1998), and given many companies the illusion of managing competence.

2 Competence issues in an organisation

We came in contact with the HR manager at a company called Iquity Systems who had realised that their employees are critical assets in their competitive knowledge-intensive marketplace. The critical assets being their employees and their competence thus realised the importance of managing competence in the company.

2.1 Iquity Systems

Iquity is a software company that develops real-time billing systems and solutions for sponsored communication for the emerging communication industry. Iquity's systems have been implemented successfully in several countries worldwide, for example Australia, Germany, France, Italy and South Africa. The company was founded in 1995. Iquity has offices in New York and Stockholm, and has grown to about 100 employees at the present.

The company got started after the deregulation of the communication market, and with the rapid development within these areas, including Telecom, Internet and so on, Iquity saw a demand and went into business. It first started out as GratisTel, and later included other services or products and changed name to Iquity. Also, as a result of the rapid technological improvement and development as well as a change in customer demands, Iquity has grown at a tremendous rate, and are now around 100 employees.

The product is the so-called IQ platform software, which enables delivery of individualised content, real-time rating of subscriber usage and individualised pricing and settlement of commercial transactions across a broad spectrum of wireless and non-wireless communications media, including telephony, Internet and broadband. The products currently used are the ones stated below:

- GratisTel, a sponsored telephony content-delivery product.
- OptPay, a real-time rating, presentment and settlement product for telephony, Internet and m-commerce.
- NicetNet, a web-based content-delivery product.

As Iquity is a software company, the software engineers are the ones actually creating the product sold. Salesmen are naturally the people travelling around the world getting the product sold. These two parts could be seen as the backbone of the organisation hence most people work within these departments. In the organisation, there are also departments such as Administration, Marketing, Human Resources, Technical writers, and Financial etc.

2.2 Assignment

In our first meeting with the HR manager, she expressed the need to identify and map the competence in the company. Reasons for this was that she saw the benefit and felt the need to identify what they know, to get a picture of the competence in the company as a base for recruiting and development activities. Also, Iquity has grown rapidly in a few years and has not had a structured HR department and has recruited and developed employees on an ad hoc basis. Since the company has grown to become a company with around a hundred employees, she feels that it is of great importance to structure the activities concerning competence.

We got the commission to identify what constitutes competence at Iquity Systems, meaning the knowledge/skills needed to accomplish work, in order to manage it in an effective manner.

2.3 The aim of this investigation

The aim of this investigation is to identify what constitutes competence at Iquity Systems. We will in this thesis look at the Software Engineers.

3 Identifying Competence at Iquity Systems

"...challenges are opportunities to improve – by exercising our attention, understanding, and ultimate creativity. This is not dissimilar to the Chinese notion of crises: two symbols meaning "danger" and "opportunity."" (Senge, 1999)

The problem of identifying competence is nothing new. In the beginning of the 20th century, Taylor was on the quest of increasing productivity by searching for the "one best way" to perform a task and then train and create competent men. This was done by his famous time and motion studies, trying to identify and describe the competence of the most efficient workers. By using the information as a base, managers could set up systematic training and development activities in order to improve the competence of the less efficient worker. This more or less resulted in standardised and routine work.

However, using Taylor's method today is highly questionable. Today, the majority of work in industrialised countries is knowledge-intensive and does not fall under the Tayloristic umbrella, tasks are often more complex and thus resist standardisation. Therefore we started to explore contemporary management approaches in identifying competence in an organisation.

3.1 Finding a Research Approach

Once we together with the HR manager had decided to identify competence in the company, intense and fruitful discussions on how we should go about it emerged. We started to discuss and explore how we could go about identifying what kind of knowledge and skills were needed in accomplishing work.

3.1.1 Measuring competence

Initially, the discussions circulated around finding measurements for different types of knowledge, attributes, which could be used to identify possible gaps in an employee's knowledge in that specific attribute. We found a large amount of such theories in Human Resource Management literature (Gael, 1988; Ferris et Al, 1990), referred to as Job Analysis, basically doing the same, identifying gaps between a worker's knowledge in specific attributes seen as crucial in being competent.

More specifically, identifying competence in contemporary management practices is to look at the attributes, what knowledge/skills that either the individual possesses, what knowledge/skills that are seen as necessary by looking at the activities that are central for accomplishing specific work, or a mix of the two. All of these approaches suggest an attribute-based view of looking at competence, thus arguing for the idea that people viewed as more competent than others possess a greater set of attributes.

We elaborated on a specific attribute, the French language, and almost directly bumped into a lot of difficulties. When trying to find measures and looking at how we could grade employees' knowledge in order to find gaps, we realised that it was not that easy. In our attempts in trying to define and judge what was a "high" or "low" level of knowledge in a specific attribute and how this could be measured made us very sceptic towards pursuing our task in this way, it felt kind of shallow.

Measurements and scales were hard to define and we asked ourselves what was going to determine a “high” grade, a 10 in a scale 1-10.

Furthermore we had a lot of unanswered questions about who shall be involved in the process of determining and judging the measurement scale and the grading of respective individual. Also, we saw problems in predefining attributes necessary to accomplish work in a competent way. Probably different persons have different opinions on what is important and not important in accomplishing work. One of the main issues was that we perhaps could miss something that an individual saw as important but that person had not been in the process of selecting attributes, consequently, we could miss out on knowledge and skills since they may not exist in our predefined picture of competence.

3.1.2 Considering the “silent knowledge”

We started to view the attribute-based perspective as a troublesome way to identify competence among the Software Engineers at Iquity. Not only the problem of finding measurements and appropriate scales, but also another issue came up in our discussions, namely something we in the beginning referred to as “silent” knowledge. We felt that using an attribute-based perspective would overlook the silent knowledge. By silent knowledge we mean knowledge that is hard to explain or present explicitly, knowledge that people use in their work but not that easy to point out. We felt that silent knowledge would be hard to define and include in an attribute-based investigation, thus would be hard to capture. At the same time we agreed that the silent knowledge was something important and interesting to try to capture in order to identify competence. The reason for it to be interesting was that we felt it could be a large part of accomplishing work, but it was really hard to put a finger on it, thus hard to articulate attributes that could be measured.

When we talked to the Chief Technical Officer at Iquity, he expressed that there is a great deal of such knowledge in a Software Engineer’s work. He compared it with painting art. It is “easy” to learn the techniques to paint art or develop software, but what distinguishes a really good painting or software from a mediocre one is very hard to explain. Describing competence in being a Software Engineer is often limited to technical knowledge such as programming languages etc., which of course is necessary knowledge, however is only a small part of being competent. He said that:

> “…to put the finger on that kind of knowledge feels almost impossible, but would be very useful if it could be expressed explicitly…”.

Having these problems in mind, we became rather critical towards the attribute-based perspective and started to search for other potential ways to identify what competence in working as a Software Engineer is.

3.2 Shift in Perspective

"I hear and I forget. I see and I believe. I do and I understand." -Confucius-

Exploring the literature made us aware of the fact that using an attribute-based perspective is to adopt a rationalistic approach in identifying competence. The criticism against using a rationalistic approach in the literature more or less confirmed the short cuts that we identified earlier, that the attribute-based approach overlooks the silent knowledge or “tacit knowledge” used in the literature. (Sandberg, 1994; Brown & Dugiud, 1991; Cook & Brown, 1999)

3.2.1 Tacit versus Explicit knowledge

The term tacit knowledge, introduced by Polanyi (1966) refers to knowledge that an individual *knows* but cannot easily explain explicitly. Tacit knowledge is intuitive, non-verbalised and not yet articulated. Targama and Diedrich (2000) state that the distinction between explicit and tacit knowledge is one of the most popular in trying to classify knowledge. Cook and Brown (1999) explain the distinction between explicit and tacit knowledge in a forceful way:

> *“…Many people who say they can ride a bicycle will claim, when asked, that they do not know which way to turn the handlebars to prevent a fall to the left or right. However, since staying upright is part of knowing how to ride a bicycle, anyone who can ride must, by definition, know which way to turn the handlebars to avoid a fall. What they can’t do is* ***say*** *which way to turn. So there’s something known by everyone who can ride that most cannot say…”*

In this example we can see that what they can say is what Polanyi called the explicit dimension of knowledge, but what is known by everyone who can ride a bicycle is what he referred to as the tacit dimension of knowledge. If we accept the idea of tacit knowledge and its characteristics, it seems hard, almost impossible to treat knowledge and workers as two separate entities. More practically, listing “tacit attributes” in a job description, or in a competence inventory list seems practically impossible in an attribute-based approach.

3.2.2 How do workers accomplish their work?

Other criticism mentioned in the literature is that the attribute-based perspective tends to predefine what competence is and thus may only confirm a researcher’s own model of competence rather than capture the worker’s competence. Also, it does not show *if* the workers use the attributes defined in their actual work or *how* they use them in performing their work. Consequently, it can be hard to explain why one of two workers identified as having the same set of attributes is performing his/her work more competently than the other one. As a result, the descriptions of competence are indirect (Sandberg, 2000).

Orr (1990) with his ethnography showed the divergence between espoused practice and actual practice. He found that service technicians at Xerox did not work according to the manuals, job descriptions and the content of the training programs. In tricky situations, service reps replaced the manuals written by experts with trial and error, ”war stories” shared with the other service reps and found solutions that did not exist in the manuals. They developed their understanding of the machines not in the training programs, but in the very conditions from which the programs separate them, the authentic activity of their daily work (Brown & Duguid, 1991). The sharing with each other is most often referred to as ”communities of practice” in which participants share experiences and ”war stories”, which helps them accomplish their daily work.

Teigland & Timlon (1998) have done a similar study of Software Programmers. In that company, management had developed and implemented several formal mechanisms to facilitate knowledge flow. However, the observations found that the Programmers did not use the formal mechanism in the way desired, instead they relied heavily on the organisation's informal organisation, communities of practice, to get the job done.

To conclude, many organisations tend to abstract training programs, manuals and formal job descriptions from actual practice, overlooking details in practice (Brown & Duguid, 1991). These studies have shown that workers, in daily work situations use other knowledge than described by the organisations, as a result, the rationalistic approach seem to overlook the "practical" aspects of competence.

3.2.3 Ontology and Epistemology

Perhaps an explanation to why we initially adopted the rationalistic approach could be that it has been the dominant epistemology in Western culture during the last centuries. Knowledge is seen as an object, "body of knowledge", and is something that individuals supposedly possess or can acquire. Knowledge is seen as object independent and beyond the individual and there is an objective reality that exists beyond the human mind.

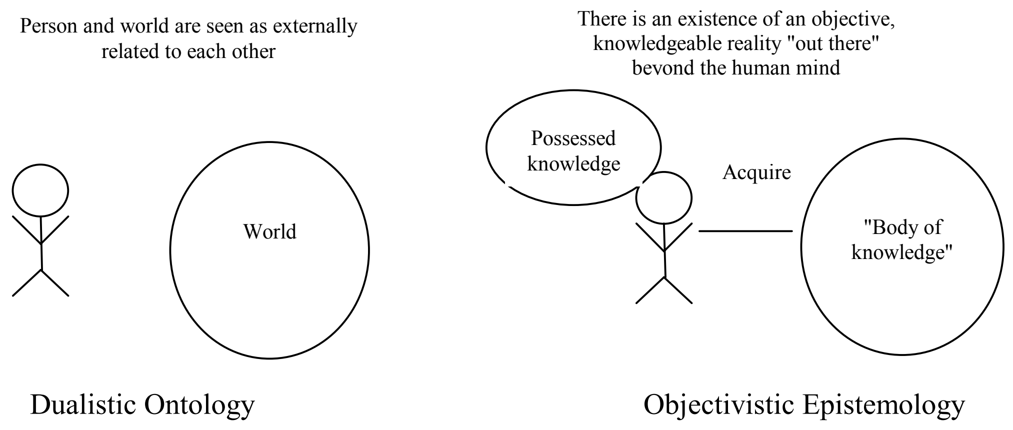
Explicit knowledge has more emphasis than tacit knowledge, tacit knowledge is something that should be made explicit in order to be understood and useful in practice. Looking at the educational systems during the last centuries, there has been a strong belief in theory first then application, thus emphasis on the transfer of attributes that then should be used in appropriate situations that appear. This can also be found in many training programs that companies use, sending employees on training courses where there is a lot of emphasis on theories or methods that can be used in specific situations.

The nature of this Western epistemology is that it views knowledge as an object, something that is beyond the individual and independent of the knowledgeable person. It views knowledge as "body of knowledge", something that a person must acquire to be knowledgeable and is something that is possessed by an individual who has acquired it. Sandberg & Targama (1998) call this objectivistic epistemology, assuming the existence of an objective reality independent of and beyond the human mind.

Cook & Brown (1999) refer to it as the epistemology of possession, where everything associated with knowledge rests on a single traditional understanding of the nature of knowledge, since it treats knowledge as something people supposedly possess.

Also, the rationalistic research tradition views person and world as two separate entities, which Sandberg & Targama (1998) refer to as dualistic ontology. Competence is therefore viewed as two separate entities, person and work, and that they can be studied and investigated separately, more specifically, competence is divided into attributes the worker possesses and a list of work activities. The objectivistic epistemology and the dualistic ontology is the explanation why, as we argued earlier, the descriptions of competence using the rationalistic approach are indirect and context-free, thus overlooking the tacit dimension of knowledge.

Figure 1 (Own construction)


Person and world are seen as externally related to each other
World
Dualistic Ontology
There is an existence of an objective, knowledgeable reality "out there" beyond the human mind
Possessed knowledge
Acquire
"Body of knowledge"
Objectivistic Epistemology

On an organisational level, Van Grogh & Roos (1995) argue that organisations have not paid considerable attention to the fundamental issues of epistemology. They continue to say that knowledge has mainly been taken for granted and treated merely as information, thus a relatively narrow picture of the dimensions of epistemology focusing on knowledge contents rather than on what organisations know or when they know.

Reflecting back on our attempts to identify attributes important in being a competent Software Engineer and then finding measures to identify gaps would have given us an indirect description of the competence in the work as a Software Engineer. Unconsciously we viewed the Software Engineer and their work as two separate entities trying to identify specific attributes that were important in accomplishing work, thus predefining what competence is, and then to measure the Software Engineers according to those predefined attributes. Consequently, our view of epistemology has been rather limited and pretty much focused on knowledge contents.

3.2.4 Knowing in action

Cook and Brown (1999), call for the attention towards an epistemology of practice, looking at knowing as action. This notion provokes advocates of knowledge as being possessed to change to the notion of knowledge as something inherent in action. The study by Orr (1990) showed that repairmen at Xerox partly replaced the explicit knowledge in manuals and training programs produced by experts that supposedly possessed “expert” knowledge, with knowing in action that the repairmen developed by sharing “war” stories and experiences with each other. The members of this so-called community of practice created their own “truth” by interacting with each other and interacting with the world. Thus, the knowledge within this community was not only some object or some attributes independent or beyond them described in a job description or attributes learned in a training course.

The notion of knowing in action provoked us to think in terms of “doing” rather than only possessing knowledge. If we take the Software Engineers at Iquity, the organisation will not fully benefit from having a Software Engineer who has a great deal of knowledge within the field, an expert, if he does not use it in his work. The knowledge is not useful for Iquity if he/she does not develop action patterns where this explicit knowledge is used. This could help to explain why two workers identified as possessing the same set of knowledge according to the rationalistic approach, can vary in how they accomplish work. One seen as more competent may have developed action patterns, which uses his knowledge in a more complete way than the other. Cook & Brown (1999) explained it like this:

> *"…An accomplished engineer may possess a great deal of sophisticated knowledge; but there are plenty of people who possess such knowledge yet do not excel as engineers. This means that if you want to understand the essentials of what accomplished engineers know, you need to look at what they* ***do*** *as well as at what they possess…"*

It is not just to see the possession of knowledge as something that enables action or practice, knowledge must be seen as a tool used in knowing. Also, improvements in work performance are not necessarily the product of acquiring more explicit knowledge, it could be a result of developing new actions patterns using already possessed knowledge.

In our case, we want to identify what constitutes competence as a Software Engineer. Without taking knowing in action into consideration a lot of potentially important issues of work might be overlooked and missed. According to Brown and Duguid (1991), knowing in action is central to understanding work, thus what constitutes competence. Furthermore, separating competence as a Software Engineer from the knowing in action, the Software Engineer is engaged in when working, distorts and obscures details of that knowing in action, consequently, competence as a Software Engineer is not fully understood.

3.3 Towards an Interpretative Approach

If we embrace the notion of knowing, we have to consider that a person engaged in "knowing" is interacting with the world. A person, the Software Engineer, interacts with a phenomenon, his/her work. Cook and Brown (1999) state that "knowing" is about relation: it is about interaction between the knower(s) and the world. To grasp the competence in working as a Software Engineer we need to understand the interaction between worker and work.

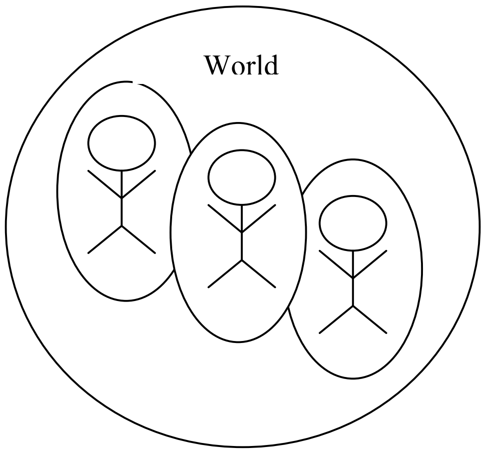
Viewing worker and work as two externally separate entities was something Weick (1979) saw as limited in positivistic approaches. He saw the need to analyse and describe the ways in which individuals enact their situations, the initial process of knowing which situations become meaningful for them, in order to understand what constitutes human action.

Husserl, the father of phenomenology, had a great interest in the lived experience; the stipulation that person and world are internally related through the persons lived experience of the world.

An alternative approach to the rationalistic approach to competence is the interpretative. In interpretative research tradition, the main feature is its phenomenological base; the stipulation that person and world are inextricably related through persons' lived experience of the world (Berger and Luckmann, 1966).

According to Norén (1995) researchers with this approach assume that there is a world, that we all share, a life-world, an everyday reality, but individuals can live with different relations with this world and make different interpretations of it.

Figure 2 (Own construction)



Person and world are inextricably related through persons' lived experience of the world

This implies that a Software Engineer and his work are not two different entities; it is one entity that is formed by the Software Engineer's lived experience of his work. As a result, competence is therefore constituted by the meaning the work takes on for the Software Engineer in his/her experience of it (Sandberg, 2000).

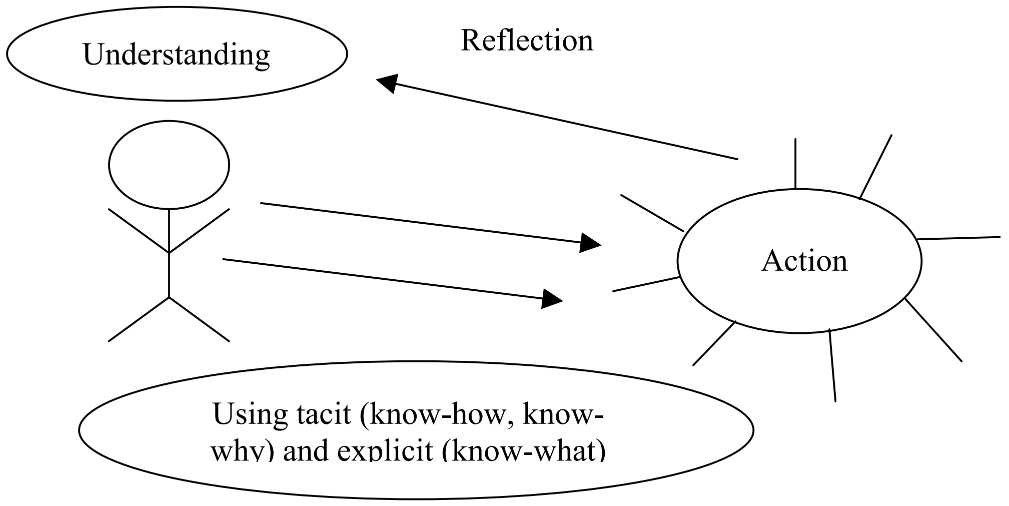
3.3.1 What shapes action?

By looking at competence from an interpretative perspective, we automatically consider the context in which the knowing occurs in contrast to rationalistic approaches where attributes in accomplishing work are seen as context-free. This enables us to consider the tacit dimension of competence, which we felt that we would not consider by identifying competence in a rationalistic manner. Referring back to Orr's study (1990) on repairmen, they developed knowledge in action which was not coherent with the explicit knowledge in the manuals and training programs, tacit knowledge that evolved through their interaction in their work and with each other. Consequently, attributes used in work are not context-free, rather context dependent through workers' ways of experiencing and making sense of that work. Therefore it seems more essential to look at workers lived experience of work when identifying competence rather than the attributes themselves.

In adopting an interpretative approach, we accept that the meaning the work takes on for the Software Engineer and his experience of it shapes his action. In other words, a person's understanding of a phenomenon shapes his action. More specifically, a worker's understanding of his work shapes his action in accomplishing work. Consequently, we could say that a Software Engineer's understanding of his work constitutes competence in his work at Iquity. This understanding works as a basis for action, allowing a repertoire of action but also limiting the action alternatives available for a worker (Targama & Diedrich, 2000).

In this action shaped by the individual's understanding, the individual uses both explicit and tacit knowledge. Even if we want to abandon the rationalistic view of knowledge that has an emphasis on explicit knowledge, we cannot disregard explicit knowledge. What we mean is that explicit knowledge is of course necessary and is one of the ingredients in action. The explicit knowledge used can either exist outside the individual in form of information of some kind or explicit knowledge already memorised or stored in the individual's understanding. As said before, even if an individual supposedly possesses some kind of explicit knowledge relevant to a given situation, it does not necessarily mean that the individual uses it in a given situation, it depends on the individual's understanding of that given situation.

Figure 3 (Own construction)



3.4 Phenomenography as an Interpretative Approach

By adopting this viewpoint, we need to find a way to investigate understandings.
In 1979, a research group in the Department of Education, University of Gothenburg, coined the word "phenomenography" (Marton, 1981). Basically, phenomenography is a method to identify people's understanding of a given phenomenon. It is concerned with the relations that exist between individuals and the world around them.
Through in-depth interviews the phenomenographer derives a person's understanding, or the word conception most commonly used in phenomenography. The term conception is used to refer to individual's ways of experiencing or making sense of their world (Sandberg, 2000). The experience of various phenomenographic investigations conducted, show that people's conception of a given phenomenon could normally be categorised in rather few qualitatively different categories. These studies have mainly been conducted within the educational arena (Marton, 1981;1986; 1997, Säljö, 1982, Pramling, 1983), but also how people conceive phenomena in everyday life such as political power (Theman, 1983). Also, as we intend to pursue, this approach has been used in business organisations.

Sandberg (1994) used the phenomenographic approach to investigate competence in engine optimisation at Volvo. In his study, he concluded that the engine optimisers could be categorised in three qualitatively different groups. He argued that competence in engine optimisation was not primarily a specific set of attributes, instead workers' knowledge and skills used in carrying out work are preceded by and based upon their conceptions of work. He suggests that the basic meaning structure of worker's conceptions of their work constitute competence.

As mentioned before, conceptions not only work as a basis for action, allowing a repertoire of action but also limit the action alternatives available for a worker. Consequently, the worker's conception also delimits possible action patterns and hence also defines what competence a worker develops and uses in performing work.

The method's aim is to describe an aspect of the world as it appears to the individual. Not to describe things as they are, nor whether or not things can be described "as they are", rather to characterise how things appear to people. After all, individuals do not simply perceive and experience, they perceive and experience things. Phenomenography provides descriptions that are relational, experimental, content-oriented and qualitative (Sherman & Webb, 1986).

3.5 Our choice of approach

Considering the discussion above, we choose to use a phenomenographic approach in identifying what constitutes competence as a Software Engineer, since we feel that this approach gives us a deep understanding of what constitutes the Software Engineers' competence. For our purposes it feels that it is an explorative and fruitful choice and we believe it to be an interesting and enlightening approach to pursue our task.

3.6 Modified aim of investigation

Our aim with this investigation is to identify what constitutes competence as a Software Engineer by taking the Software Engineers' conception of their work as the point of departure using a phenomenographic approach. Also, what we have partly done already is to identify a potential other way to view competence in organisations other than the attribute based.

3.7 Identifying Software Engineers' conception

The starting point for us was to get to know the Software Engineers and get a picture of their work. We introduced ourselves during one of their Monday meetings and gave them a presentation about what we wanted to do. We cleared out some questions around our intended investigation and booked interviews with most of them. They are in total eleven Software Engineers working at Iquity, two of them did not want to be interviewed and one of them did not have the time, consequently we had eight interviewees.

Since our purpose of the interviews was to get the Software Engineers' conception of their work we decided to asked them only four questions, which we intended to follow up with appropriate questions. With appropriate we mean follow up questions that are directly relevant in the context the interviewee is talking about. Using a phenomenographic approach, we wanted them to describe their experience and interpretation of their work by having as open-ended questions as possible to make them speak as freely as possible. Also, important in a phenomenographic approach is not to steer or influence the interviewee during the interview in any way. Our goal with the interviews was to identify what constitutes competence for the Software engineers at Iquity Systems. The four questions were:

- What does working as a software engineer mean for you?

Purpose of question: To identify the Software Engineer's conception of his job, what they think is the task of a Software Engineer

- What is a competent software engineer for you?

Purpose of question: What is seen as competence in this job

Two additional questions were asked to get ideas from the interviewees about how they learn, and what could be done in order to enhance competence development activities.

- How do you acquire competence needed when working as a software engineer?

Purpose of question: See how software engineers procure competence.

- How would you go about sharing your competence?

Purpose of question: How the competence is shared with others.

Before we conducted the interviews, we carried out a pilot interview with an employee who formerly had been working as a Software Engineer. The purpose was to test the questions and to try out interviewing under phenomenographic conditions. For a more detailed description of our method, see Appendix 1.

4 Software engineers' competence

4.1 What is software engineering?

George, who is a student, sits down in front of his computer to do his assignment he got the other day. He is to write a 5 page short report on the benefits and drawbacks of the Balance Scorecard, which is due soon. He turns on the computer, and sees on the screen a big logo that shows: Windows 2000. After a few seconds the computer is done computing, meaning it is done with the start up sequences. He, with the use of the mouse, presses the *Start* button, and gets up a menu, where he goes up to *Programs*, where another menu pops up, and he goes to Microsoft Word as it is the program he is going to use today. When he is in the program, he goes up to *File*, then presses *Open*, and finds the document, "Balance Scorecard report"*,* which he started last night and hade *Saved*, and double clicks on the document and it opens. He goes to the end of the document and starts typing the rest of the report, and after a couple of hours he is done. He goes to the top of the document and presses *Tools*, and then *Spelling and Grammar,* the computer asks him if he wants to correct the word, and gives suggestions for spelling corrections. When he is done with that, he presses *Edit,* goes down to *Select* All. Then he goes into *Format*, selects *Paragraphs*, because he wants the text to be double spaced, he selects that…

What George probably never reflects about is that people have actually sat down, figured out how we can make this as easy to use as possible for different users, meaning a traditional typewriter with a lot of extra functions. The people responsible for creating Microsoft Word and all its functionalities etc. are software engineers. A software engineer is the one developing all this, what we can call software, or a system, which is probably made up of several software.

When we, meaning people outside the software engineering profession, try to understand what software engineers do, we usually think that they are programmers. As it is rather new for people and not many of us really know what a software engineer is, we cannot really relate to it as we can in other professions such as being an accountant or a construction worker. A software engineer expressed this statement concerning his work:

I If you were to explain for us who do not know so much, what is a competent software engineer?

S Yes, because it is like this, that when people ask me: *so what do you work with*? Then you say that you are a software engineer. Then they say, *yes, a programmer right!* And the difference between… meaning to explain what else there is than just sitting and programming, that is really hard, I think. There is no one who really understands what you are doing. The only things you can say, is that I have devised that system, Gratistel for example, but they cannot [grasp it]. It is very hard to show what you have done. Explaining that we have done that thing, but you cannot explain, what is complex and what is hard. As a teacher it is very easy to say, *yes this is that*. In many other professions, people can relate because… if you are an economist, then people can understand it, because they hear it on TV, or they take care of their own economy, and they have it at work etc. Here it really is… *oh, yes, a* *programmer,* but that is actually very distant from what it actually is that we are doing, because programming is what I did in the in the start or in the middle of the 80s/…/

He continues to say,

S /…/ And it is so much more, and that is…by just looking at the time spent on the actual coding (programming) here, it is probably a very small part of everything that is done/.../

4.1.1 Teamwork

Also, what is important to mention is that a Software engineer, like many other professions, is teamwork. This is very important to state this, as when they work in teams, they do not have responsibilities to only themselves, but also to others. This will be discussed later in the paper, the importance of it. This quote shows how it used to be more individual work, but has become more than just that:

I Is there something else that you want to say, what you see, for you, as a competent software engineer?

S No, not really… Another thing that is very important in software engineering, it is not work that you do alone, more and more. Maybe ten or fifteen years ago, you could sit and write a small software program yourself, and still there are some geniuses who are doing that, but at my level, my job is done as teamwork.

4.1.2 The task

A software engineer at Iquity Systems Inc. is someone who is making a solution mainly through developing software, but also in his task is the maintaining of old software, meaning software or systems already created. It is a craft one could say. The way it is done at Iquity, also recommended by theorists (Berztiss, 1996), is that they are following a so-called development process, where roughly in general four parts are included: requirement specification, design, implementation (coding or programming), and finally testing. McConnell (1993) compares it to a food chain, "In an ecologically sound environment seagulls eat fresh salmon. That's nourishing to them because the salmon ate fresh herring, and they in turn ate fresh waterbugs. The result is a healthy food chain. In programming, if you have healthy food at each stage in the food chain, the result is healthy code written by happy programmers".

In the software engineer profession, there is not a universal way of doing it, but rather each organisation itself finds methods and processes how they decide to work: This is what is said by one of the software engineers at Iquity:

S But, it is to know the methods, and there is a problem with this, if you, for example, compare computer science with civil engineering. If you are going to build a bridge, there are ways to build it, there are standards, there are measuring methods like "solid mechanics", which state that this will withstand, but of course they still make mistakes. But, within computer science there are no standards. *There are methods, but no direct ones that say,* *this works and this does not work, rather one that eliminates possibilites through trial & error*. There are ways of doing it, so that it will be good, and later to testing (final part in the development process) it so in the end... one has tested that it should work. I mean, you use that flora of methods and techniques, and such that exists. So, it is not about knowing a lot of programming languages and such things, because that is a level deeper. There are a lot of programmers that are really good at programming, but they do not understand the whole (development process).

In the requirement specification phase, the objective is to gather and pick out the requirements, set by the customer or someone else, which are the "good" requirements to solve the problem. In the design phase, the software engineers design how it looks, its structure, what does what in the program. The implementation phase is the actual coding or what we call programming, and that is realising the design, meaning filling in the characters so that the program does what it should do. Lastly, there is the testing phase where one checks the code to see if there are any bugs, meaning, is the system doing what we intended it to do. A typical program design could look like this:

S But, if someone says to me, or to you by all means, write an application software that reads user name, saves it on a database, and then translates it to Japanese/…/ /…/ then I am going to have three components, one translator, something that can read over the net, and something that connects to the database, yes there I have three parts. How am I supposed to solve the thing with the database? Yes, then I have one part of a problem, and I work with the database, then Ok…this one I am able to buy, and I get a SQL database, learn SQL so I know it - so I know how to search on a database. Then I have solved that problem with the database. When I am doing a Web interface, for example over the net. Ok, then I maybe have to learn a little ASP, I have to learn some language like Java, which suits. I need some kind of design, so I need to know how design works, and then I divide up. Ok, I have a design, how should I do this. Then I need an interface - what does an interphase look like, how will I use it…write in things, then I need that, then I divide that, and finally I have a design over the interface towards people, which hopefully is very well written, if you are competent enough.

To conclude what the software engineers do in their profession, or what it actually entails, is something we have taken and interpreted from the interviews we gathered from the software engineers at Iquity Systems Inc.

4.2 Three conceptions of competence in software engineer

According to our interpretation of the interviews that we gathered and transcribed concerning each software engineer's description of their interpretation of their experience, there are three qualitatively different conceptions of competence as a software engineer. What the reader should remember is that our aim was not to find an objective truth, rather to make interpretations of the interviews. The three conceptions are:

A. Competence in developing software by focusing and relying on the process

B. Competence by continuously looking for ways to improve the way one works, meaning the process

C. Competence by understanding and getting a more complete picture of the problem when developing the software

How these conceptions differ from each other is how the software engineers conceive their work, and according to these conceptions is the way they actually work. Sandberg (1994) explains it as these conceptions "delimit and organise" the software engineer's work into three qualitatively different ways.

The software engineer communicating Conception A solves the problem or develops software by following the development process, which in this case Iquity has decided to use, but also taught in school and recommended by other software engineers. In the development process, one has a step by step guideline, but it is not a straight requirement to testing kind of process, but one can loop around, and frequently go back and correct mistakes made in the former parts. But, still the problem is more or less applied to the development process.

In Conception B, the software engineers also see the process as the way software is developed, similarly to Conception A. But, what is different in how they conceive their work is that their focus is constantly on *how* one can improve the process or parts of the process, always looking for parts to be improved and refined. A question frequently asked is also *why* do we do it in this way.

Software engineers expressing Conception C are like Conception A and B using the process as it is the method of choice at Iquity Systems (see Conception C under Further Analysis). But unlike Concept A and B, these software engineers do not try to apply the problem to the process, but the process to the problem. The process becomes more of a guide or a helpful "tool" in order to create the software. The emphasis is more on what is the actual problem and how it is a problem. The workers expressing this conception are very careful to *think* of most of the things around the problem before going in to, what they call "details", and using the process as a guide.

To illustrate that there are software engineers that can be identified by this from what they have said, we made this chart. It shows that four people expressed Conception A, two Conception B, and finally two expressed Conception C. To keep in mind is that two software engineers of the four , in Conception A, could be in a transition phase, between Conception A and C, or more clearly state that they are interpreted as expressing both Conception A and C. To clarify this, in discussions we tried to understand why they were able to express, in the interviews, something that we interpreted as both categorised into Conception A and C. The reason could be if we speculated that they are in a transition phase. These software engineers are probably about to change their way of accomplishing their task, hence are seen as "confusing", expressing both Conception A and C. The other software engineers, on the other hand, were more straightforward. One can then say that what they expressed in the interviews could really be positioned between A and C, but we thought as they expressed more aspects categorising Conception A, they were positioned there. Again, what is important to notice is not really in what conception category each worker has been positioned, but rather the Conceptions themselves representing that we interpreted as three qualitatively different competencies that are of importance.

Table 1 (Own construction)

	Conception A	Conception B	Conception C
Software Engineers#	4 *	2	2

* 2 could be identified to be in a transition phase

As Sandberg (1994), we, for clarity and our purpose, decided to take the analysis a little bit further to see more in detail how they differ and are similar. Under each conception we could identify different important aspects that "delimit and organise" the software engineers way of conceiving their work, hence form the qualitatively different conceptions. We are to show them below.

4.2.1 Conception A: Competence in developing software by focusing and relying on the process

The people expressing this conception are focusing and relying on the process that is made up for the software engineers at Iquity. As their main task is to solve a problem by developing software, one can notice that they, both explicitly and implicitly from the interpretations of the interviews, are going through the process step-by-step to solve this problem. One can say that the main focus is not on solving the problem itself, but rather on how the problems can be applied to the process to get a solution. The software engineer expressing this conception saw progression of the process as central.

4.2.1.1 Solving problems by following the process

What we mean by this is that the software engineers have a problem, and look for a natural step-by-step plan provided by the process. One software engineer expressed this about what he conceived as work as a software engineer, in the interviews we gathered.

I What does work as a Software engineer mean for you?

S It is someone that can develop software, meaning not only do the coding, or whatever one says, one can write code just like that, but also design a system. Meaning simply an engineer, but it is hard to explain…

I If you say, the profession software engineer, what does it mean for you?

S It means that one knows, to start how to design, or you participate in the design. That I consider a software developer should do. One should not have specialized designers, specialized coders, that I do not believe in. But, I believe that a software developer, he develops software from beginning to the end.

I What do you mean by beginning to the end?

S You begin with, I mean everything...begin from requirements so to speak, start you design that of course generates more requirements. Starts the coding, which perhaps generates even more requirements, because you see things that you should have seen in the beginning. Test, as we do here, we are doing something that is called unit test, i.e. that you test that component that you are responsible for, before you hand it over to the systems testing. Then you get feedback from the systems testing on bugs and problems. And then you should participate in attending them also so to say. So, I think this is the development cycle for a software engineer/.../

He expresses that being a software engineer, you should not only know the different steps or parts of the process, but you should know the whole process, as this is the way you solve the problem, thus develop the program. It seemed important that knowing just one or some parts in the process, you are not considered competent at Iquity. Another software engineer communicated this how to solve problems:

S Yes, but design is when you…like here, we have something called feature design, where you describe how it should look, how you will be implementing. Implementation is the actual coding so to say. Then all the parts around, like testing and more that you also have to do.

Later in the interview the software engineer, stating the first quote under this conception, expressed this to support our interpretation of him, expressing Conception A. Following a technique, often set by the organisation, is how you should go about solving a problem according to the software engineers expressing this conception. This shows that there is a focus from a detailed level and from there; you work your way up through the levels.

I If somebody is a competent software engineer, how does he perform?

S You mean that you get a task, you should do something that does this, which is answering like this, if I tell it. Personally, you start from the technologies you are familiar with, or I do that. You immediately begin to think in those terms, how can I solve this, and then you have like Lego bits. You have the technologies you know, and eventually if you are broad-minded enough, or there is something that you are not sure about, but you know that it exists and have a hint what it is about. Therefore, you start to think, this I should be able to solve by using this programming language, for example. Like here, it is very steered also, what you should use, but that is because everything is made and based on C++ on a Windows platform and in DCOM. So, it is natural that you choose these terms to think in, how can I solve this with C++, COM, on a Windows platform.

Above statement, shows that at Iquity Systems, they have decided that C++ on a Windows platform using DCOM is the best technique for their types of problems they get. Therefore, from thinking how the problem can be applied to the process, one also thinks how the problem can be applied to the technique. A reason given is that for the problems they get, these techniques generally would work.

The software engineers also expressed the importance of knowing specific parts of the process, and of course the whole process, but this is what a software engineer expressed when we wanted him to elaborate on how he means with programming.

I But the central part you think is?

S Programming

I Programming?!

S It is what I see when you mean the word software engineer, so to say. But again, at work then you often do other things too.

I But with programming, you mean coding, right?

S Yes, that is coding, <u>the code, from principally no requirements, to given requirements, write a design, execute the design and document what you have done and show that it can be tested</u>. That is according to me, coding. The actual writing, get loops and things…sure, that is the central part, but to get there, you have to do the other parts also. So, everything is writing code, I think.

This software engineer expressed that everything is coding as everything leads to the coding as that is seen as developing software.

Another software engineer communicated this as the Waterfall principle, which is typically referred to as a logical step by software engineers expressing this conception, as when errors occur, the errors are more or less "serious" the further down in the process you are.

I You talk about all this design something. Is there anything central?

S Yes, it is like this. In the requirement collection phase you get wrong requirements, and if you do not discover that until late, then it is going to take much, much more time to fix such things and that is a lot. <u>The so-called waterfall model, you have requirements, design, coding and testing</u>…

No, but...to get <u>wrong requirements means a bigger impact on</u>…or, yes, if you do not discover them until late [in the process]. That has to do with, I mean <u>if your coding is wrong and you test it, you find out that you have coded wrong</u>. But, when <u>your design is wrong and discover that, then you have to go all the way back to the design and then do the code again</u>. But, if you have <u>done the requirements wrong, then you have to go all the way back to the requirements, redo the design, redo the coding, redo the testing</u>, so this is something everybody knows, or this is something everybody knows when you are a software engineer, or a programmer, or a coder or whatever you are. But, this is how it is. It is required to get a better design, than getting a good code, because good code you can get by testing… But, that is not the case either; you need good people in all the places [in the process]. But, that is something; you usually put new people to code some and testing and then, code some more, and then design and so on…

Software engineers expressing this conception use the process as a way to solve problems, and it is that the process itself is more important and focused upon, than the problem itself. The process itself is roughly made up of four stages, requirement specification, design, implementation (coding), and testing. They express in Conception A that you have to know all the stages, because that is needed at Iquity Systems, even if in other places they have special designers etc.

4.2.1.2 Reflecting back to where you, or someone, else solved a similar problem to be able to use a similar process

The reflecting back is done to take shortcuts to see how you or other software engineers solved the problems before you, using the process. This, of course, is very time efficient, as you do not have to go through the separate steps in detail, meaning coming up with a good design, as there already exists one before. A software engineer expressed this.

S Yes, you learn at work without thinking about it. There is so much which you do not even [think about] Now I am going to do this to be able to learn, but rather there are just things that happen around you, which you learn at an unconscious level. It is just work experience. But, right now for me, it is looking at our system and looking at the codes, and seeing how they have been done, and trying to figure out how they were thinking and why they thought in that way/…/

He continues to state this to show how he will be able to be more efficient.

S /…/It is just working with the product you are doing so you learn. That is happening more or less quickly. Simply, through observing others and what they have done, how they did it and why. All designs and to learn…it is like this that you learn/.../

Another software engineer expressed this concerning reflecting back and using the same process or method to solve the next upcoming problem.

S Yes, but a good software engineer should know all areas, because they should understand testing fairly well, and coding, design and handling of requirements, and be good in all these areas. And this, when a new problem occurs, then you should be able to refer back, because often you have solved a similar problem and can see similarities and likenesses so it is very easy. We do like this and like this, and it is a standard this and this.

This shows that there is a constant reflecting back to what one or others have done previously to be able to solve the next problem in a similar manner. Why one does this is, as explained before, in order to save effort and time. As one who uses the process, there are steps on how to develop a program.

4.2.1.3 Learning specific parts of the process

Worker expressing Conception A also states that the learning is to learn the different parts of the process, e.g. language or perhaps a technique that will be used later on.

This is very tied to the reflecting back, but we interpreted it as a separate aspect as they focus on specific parts of the process. When asked how you learn, a software engineer expressed this.

I How do you acquire the competence that you consider needed as a software engineer?

S <u>I read magazines, search on the net</u>, and if there is something else that I want to learn, DCOM for example, that you want to learn a little more in depth where it is not enough to read an article on the bus on your way home, then you can attend a course, which I know you can do here, if it is relevant for the work so to speak. You can of course purchase books, and then you can sit down and, <u>if you want to learn a new programming language or some kind of tool</u> or something, you have to <u>sit and work at the computer and play around, that is the best way actually</u>.

4.2.1.4 Interest in specific parts of the process, for example technology, in order to be proactive.

To be interested is something that the software engineers communicated in Conception A. By interest they saw it, again together with learning, being proactive, to be prepared for future needs in the process, if for example in the implementation phase, they decide to use Java instead because it is something that has to do with the Web. A software engineer expressed this about interest:

S /…/And then it can also be a little on the contrary also that <u>you try to keep up with what is happening within the area [software engineer], various areas and try to read some, attend courses here and there, or some seminar, or anything to just have that little knowledge about an area, because when a problem pops up you know that you can solve it by using this new technique or this new [other things]</u>. So, this is <u>to be reactive or proactive</u>, but that depends much on the place you work at also, because I mean in this place it is very much reactive/.../

Another software engineer expressed this about interest.

S /.../That is also a <u>good competence thing, I think, that you are interested in new techniques, you are interested in reading and such things</u>, to acquire it/…/

Later in the interview, he states this about interest and the profession as a software engineer.

S Yes, to keep up-to-date, there are so many new things happening all the time, so you have to try to keep up a little, <u>so you are able to make the right technical choices when it really is important</u>…you simply know what exists

Being interested and to get an overview of what different techniques and languages out there is important for software engineers expressing Conception A, as they are preparing what perhaps could the future needs and demands. When things then directly are of interest for the job itself that they are doing, then they know what it is about and can learn more about it in depth to be able to use it.

4.2.1.5 Summary

These aspects we interpreted as being important and in focus when the software engineer expressed this conception in the interviews. As their focus and importance was more or less placed on the process instead of the problem itself, all their respective efforts were focused on parts of the process or the whole process. The process starts with the requirement specification and ends with the testing, before it is to be developed software. In each stage, there is also a smaller guide for what to think of, so it can be said to have smaller processes within different stages. Relying on the process is something that is "best practice" for the organization as they have decided it to be in this way, as well as this is the way they are taught in school, to use a process to solve problems.

4.2.2 Conception B: Competence by continuously looking for ways to improve the way one works, meaning the process

Workers in this conception are also, like Conception A, relying and focusing on the process when they are solving problems, meaning when they are constructing software. But, what is qualitatively different from software engineers expressing conception A, is that they are always asking the question why and how things are solved, meaning they are reflecting about the process itself. They do this in order to improve and refine the actual process by which the software engineers at Iquity work and are abiding by. They are also doing this refinement and improvement individually, so that they are working better and more efficiently.

4.2.2.1 Solving problems by relying on the process

We concluded in our interpretations of the interviews that they are still relying on the process for solving problems. They are going through the process step by step in order to develop the software meaning from requirement specification to testing. A software engineer at Iquity expresses this about the process:

S /.../It is from <u>requirements collection somewhere, you start</u>, and then you <u>finish somewhere, either before the test or one includes the test also</u>. It is a pretty narrow comprehension. But, everything that is in that process, at least, is about software engineering. <u>To develop software is just that, which is in the process.</u>

He also states the following about the process:

I You talked about an engineering kind of way, how do you mean?

S /.../Yes, one should not have to hesitate whether it is really in the end going to be a bridge, or whether the bridge will bear. It will be a bridge that bears cars. It is like this, <u>because you have a process from beginning to end, so to speak. That is an engineering kind of way</u>.

Another software engineer stated the following about the process and how he looked at it.

S Yes, software engineering is about the developing processes, how one works so there is a wide variety of different theories about different development processes, but they all are pretty much about the same thing, that you start with some kind of analytical phase, where you collect requirements, then you continue on to some kind of designing phase where you from the requirements, design how you should solve the problem. When one has concluded how one should design it, you continue on to the implementation phase where one is coding hence implementing the things. And then you go to the testing phase, where you test that everything works as you intended it to do. These four phases usually are common in all development processes. So some kind of basic knowledge, how one should proceed, what is thought of in the different phases, and what type of work is needed, before you are finished with a project so to speak, is that knowledge that you need, I think.

The software engineers communicating this conception are relying and focusing on the process as the way to solve the problem, meaning developing software. Again, the process is a step-by-step, how to go about planning, how to solve the problem and it is applied to all types of problems that the software engineer faces at Iquity. But, what distinguishes Conception B from Conception A is the reflection on why things are done the way they are and how it can be made better. We interpreted this as key the key aspect or central for software engineers expressing this conception.

4.2.2.2 Reflecting over why and how things are done the way they are

A software engineer expressed this concerning the work at Iquity, and it has to do with reflecting over the way one actually works:

S It is one that sees his job in a quite an engineering kind of way. This one I think is competent, and what I mean is in being systematic about how one works and trying to improve one's own way of working and improve the group's-, and the organisation's way of working.

He also stated this about reflecting over one's work and being systematic in one's way of improving and refining the way the software engineers work in the process.

I You were talking about systematic, how do you mean?

S Yes systematically, it is from the beginning to end that you think through what you are doing first and why. That I do later, and then and why and these things, and how could you do instead. Not to get stuck on some, in some detail, a certain particular part of the development process, that this thing you can do in this way instead, because you have read an article and therefore that you have gotten an idea about something. But to be systematic in such work, it is called process development/…/

The person still sees the process as how you solve problems, but one always has to understand why and how one is doing to be able to improve the process.

I Find better ways to work, how do you mean?

S That is really the whole process that you work with all parts. If you do documentation, <u>why you write it, and what was the purpose with this documentation, and if you do not write documentation, why you didn't and in what way</u>. It is about thinking, what and why, evaluate your own work, or that way you are doing it in, not the result, but the way you got there/…/

This software engineer expressed the following statement about reflecting over how one works in the process.

S /.../You can keep up by reading more theories or programming languages or courses or whatever. Then at work or when you play around, new problems occur . <u>You learn to solve one task once, and the next time you bump into something similar, then you feel familiar and know what you did last time, and what was good and what was bad and such things that occur.</u>

This software engineer expressed that the reflecting back is done in order to see what went wrong and what was good to be able to know the next time you get something similar, you can improve yourself and the work process, which leads into the next aspect or continuously improving and refining.

4.2.2.3 Continuously improve the work process and method

The reflecting over the process and why and how one works in the way they do is in order to continuously improve the process the software engineers are working according to. This improving or refining of the process is done all the time, which is expressed by this software engineer, which also goes hand in hand what they talked about being systematic in their work.

S You <u>always have to concentrate yourself to make to whole process faster and to make it better.</u> And then you have to do the most important things first. Have <u>arguments why and what is most important and that is systematic - to arrive at such arguments</u>, this here is the most important that we have to do. There are, by the way, complete methods to get you from total chaos to a totally controlled development process, there are complete models for that. This you should fix first, then you should do this, and later you should do this. It is like a meta-process, a process for process improvements, there are such things also.

This person also mentions that there are organisations, which he identifies, that are working with the improvements of processes, so he can put what he is doing in relation to something, and this is shown by this quote:

S There is a company like <u>Rational for example; that in principally exclusively only are making tools that help the development, in the development process</u>. They are selling a method, which you can work according to, and they sell tools that support this method. There are several companies, but Rational is the largest and the most famous, I think.

Another software engineer expressed this about continuously improving or refining the way they work:

S /…/So there is one part and the other part is that we ourselves have realised that this doesn't work, we have to improve ourselves in this, this we could do, instead. Put some demands on ourselves next time we do something, we shall include these parts also/…/

These workers, expressing Conception B, see the continuous improvement and refining of the way the software engineers' work as very important and this is the way they work, which both steers their action, but also delimits them. They are looking at specific parts of the process and the whole process to see if this can be made more efficient, meaning smoother and faster steps through the process. This is done systematically, trying to find problems, but also by bumping into them during the development process in the job, thus fixing them on an ad hoc basis or remembering for next time when they face something similar.

4.2.2.4 Knowing how to work together

To know how to improve and what to reflect over the software engineers expressing this conception see ways of how to work together and how to improve this. Different from Conception A is that the software engineers in Conception B communicate the refining of teamwork, meaning how they work together. A software engineer expressed this about having a plan so that each team member, hence the organization knows how to work together.

S /…/But, the goal is really a totally controlled process, where you are all the time, given the requirements You always know where you are, how the situation is right now, [this] is something you always know when you are at CMM level 5[1]. And you know exactly where you are going and you have a complete plan to get there. And, then you don't know if it is going to be like that, but you have a complete plan for this. This plan is of course adapted all the time, controlled yet adaptable, because that is some kind of ideal [controlled process] then, which I agree with, that is the best.

Above the software engineer expressed the need to follow process as a plan, so that you and the others know where you are in the process, but as the focus always is on how one can improve the process, as it is not absolute, a process can always be made better.

Another software engineer expressing Conception B said this in the interview:

S Then I think that it is very important that you work together with other people, to cooperate, because you usually work in project groups, and a competent software engineer should know that it is faster and smoother to solve a task and it is better solved if you communicate with each other and share your own experiences, that you know the importance to follow a development process. If you have ten people that are going to solve a task, and in some way the others are depending on what I am doing all the time. If I don't put any effort inti my

[1] Capability Maturity Model set up by Carnegie Mellon University

part of the task, then the others are going to suffer. A competent software engineer should know *how* you should act when you work.

The how is in focus for software engineers communicating this conception. You should know how to work in a team, still following the development process. As teamwork (see section 4.1.1) is necessary for their profession, because one person is not able to do it all usually, but then there is a dependency on each other, and to get the teamwork running as smoothly as possible.

4.2.2.5 Learning how things work and why

An additional aspect for the people expressing Conception B is that one also has to learn how things work so that one can find better ways of working with the process. This is tied up very much with the reflecting aspect of Conception B and this is communicated by this software engineer in one of the interviews concerning learning other things, as well as internally knowing how things can be done better:

S /…/There are a lot books written on how you should work, how you should make a system, how you should do programming so that it works. I have seen those with an academic background often find it easier to attain such knowledge. To buy these books, to read these books is very important.

But, it is not only learning from outside sources, but one of the most important ways you learn as said above, is by thinking about what you are doing.

S /…/There is material on the web, that you can read, but I still try to learn all the time from what I am doing. To think, now I am doing this, now I am solving a particular problem in this way, how have I solved this problem earlier? Then I did it in this way and what problem did that bring when doing it like that. Yes, then it became bad in this and with this way, you solve this in this manner. Meaning, to try to think about what you are doing, so to speak, and why and how you proceeded, reflecting over it.

Learning how things work and why is done by both looking at outside sources such as books and on the web, but it is also very important that you experience it yourself and understand what went wrong or what was good in order to make it better. This is also linked to the next aspect of having an interest in how to make things better.

4.2.2.6 Interest in identifying how things can be made better.

The software engineers expressing this conception has a great interest in trying to find better methods on how to work, meaning, at Iquity, improving the process. A software engineer expressed this about being a competent software engineer.

I What is a competent software engineer according to you?

S To start, I think it is important that a software engineer is well educated, for example civil engineering, or computer science or something. Because, there is a lot of background knowledge and theories that you simply have to get by reading. And, if a person has been interested in computers all his life and has coded and everything, then you go down your own path, you learn things that

> you have to learn to be able to get where you want, but you never think of learning these other things that you directly don't see the usefulness of, <u>but still it is of help to have some kind of clue how things are working and why</u>/.../

An understanding for how things are working and why is something that the software engineers in conception B, interpreted by us, communicated in the interviews. This also is linked to how they learn, where they constantly are reflecting how and why they work the way they do to be able to improve and refine.

4.2.2.7 Summary

To summarize this conception, the software engineers expressed that there is a process that they have decided to work with in solving problems, hence developing software. The reliance on the process is similar to people expressing Conception A and steers but also delimits how they work. But, instead of just relying and focusing on the process, the software engineers communicating Conception B, tried to see how and why it was done the way it is done in the process, to be able to find better ways to work, to be faster and as efficient as possible. A lot, or everything nowadays, is done in teams; a focus for the process refinement is to get the team to work as well as possible. As their focus is on the process improvement and refinement, they also learn things and are interested in how they continuously can make the work process better, and it seems from the interviews that they are constantly looking at what other parts of processes look like, as frames of references, but also they are kind of running into things as they work that they can improve and refine for something that they face later.

4.2.3 Conception C: Competence by understanding and getting a more complete picture of the problem when developing the software

The software engineers communicating conception C are also using a process when they are solving their problems by developing the software as decided by Iquity Systems. But instead of looking how the software engineers can apply the problem to the process or continuously find more improvements to the process, these software engineers are looking at what, how and why the problem actually is a problem. To further explain this is that these workers focus on what the problem is, and the process could be seen as a guide or help "tool" to solve and develop the program. The emphasis is how can one get a well thought picture of the problem as possible before looking at how to solve it. Important to state, is that we do not say that the software engineers communicating this conception are not using the process, but on the contrary use it, but see the meaning of using it for that specific problem at hand.

What is central in this conception is the way of thinking and analysing that the software engineers have, to see what is the problem and how it is a problem, to get the most optimal solution to the problem possible.

4.2.3.1 Problem focused approach

This concerns a more analytical way of thinking before actually attempting, by in Iquity's case using the process, to solve the problem, which is creating or developing software. The reason for this is that the software engineers, working like this, want to find out what the problem is, being conscious of the aspects of the problem meaning what will be the outcome. To put it in easier terms; what do we (software engineers) want the solution to do, meaning the aim of it.

I What do you mean with this basic knowledge?

S Yes, you have to know how you, given a problem, something that should be solved, then down to the program code and all those steps of thinking between. Partly in terms of how one really sees it and to find out what the customer wants and to write it down…because usually the customers don't really know what they want They want something that will make their life simpler in some kind of way Like an ordering system or something like that. And they maybe don't know exactly how it should work when it is computerized. But they know that they usually write like this in the book and then they write like this also.

The software engineer continues saying:

S /.../Me myself, I usually try to think - divide it up in abstraction levels so that you can describe it on a "higher level". To do this system on a higher level, where you do not include the details further down. And then, refine it until you finally get down to the code/…/

Another software engineer expressed a similar statement concerning the same kind of thinking and analysing.

S /…/So software engineering is all about thinking and analysing. Once you analyse, the first step actually is what you are doing in software engineering - you are analysing. After that you choose a tool - use the tools to realize what to have to solve the problem. So, it is two parts, the way of thinking, and using tools to realize what you thought. This is quite general, but what is very common in all different areas of software engineering is thinking and that is actually the problem, now comparing to 10 to 15 years ago, because there are a lot of new tools and people lost by using tools, before thinking about the problem. This became the normal way of thinking, if I ask them how to use these tools, they say ok: I know and I will do that - but I say what are going to do/.../

Above, the software engineers are stating that there is a problem that someone has and the software engineer's aim is to see what and how this is a problem. Before you go into and use the process to solve the problem, if this is the method of choice, then you try to get an overview, think in abstract terms, to see what is involved and how they perhaps can influence the solution. The software engineer tries to describe and understand the problem on a "higher level." This is quite different from the software engineers expressing Conception A and B, as they are focusing more on a detailed level, meaning going from the technique chosen, and then work you way up the levels.

4.2.3.2 A clearer picture of how to solve the problem

Tied to what we said above about the focus being on the problem itself, which means that the software engineers focus their efforts to get a good overview, hence as clear a picture as possible, to how to solve the problem in the beginning.

S /…/And that is also a modification, because sometimes you see when you are up [in the abstraction level], then you see - this is going to be hard, this technique we do not really control, down here. Then you go further down directly, we try - *yes it works*, and you go down [in abstraction levels] and work some down there, go up a little on a higher level, and then, *now I am satisfied* - now I can continue down here from this higher level. You are simply risk identifying. You do some part that is hard, and then you continue. Divide into large parts, look - here we have found something difficult, and then we do that. Because then you are taming the risks, so that things do not take too long /.../

The software engineer continues to say.

S /.../Finally you get a component through making the problem simpler and simpler to solve all the time. You solve it as easily as possible and then you get problem parts, then you solve the problem parts. Then you take shortcuts, by sometimes going down and up, if you feel that it will be risky somewhere. And that is some kind of intuition I think, that you feel that it is some kind of risk - this seems to be complex you feel.

The software engineer sees that further down the road there will be problems. But, if one can see the risks in the beginning, there will be a smoother way to complete the software, expressed below.

S /.../And later, all the mistakes are going to drop in afterwards, but that is perhaps something that you can afford sometimes. But, often you cannot correct something that has too many mistakes in it, meaning that you cannot remove the mistakes or make something that is bad, correcting the mistakes, so you get something of quality, but there is very much doing the right things in the beginning which applies to software.

Another software engineer expressed the following statement

S /…/These parameters, some of them, you can find. There are a lot of parameters. Some of these parameters are so small we can't find them. We can't point them out easily. Someone with experience has all these parameters, so his decision is based on all this, because of the experience. All of these parameters he uses in his decisions. All of these are small things, which are not conscious. The sailor looks at the clouds, the sun there, the colour of the water, and all of those parameters, and he takes a decision, and he doesn't know how, but he has this feeling.

The software engineer expressing this conception states that the more experience you have, the more parameters you can identify, meaning that you will have a better overview of the actual problem and how to solve it. Also, when the software engineers are “risk identifying” in the beginning, he will more smoothly solve the problem, instead of bumping into risks being forced to go back and forth in the process. The software engineer continues.

I I think you said before, with experience, that you learn more parameters?

S Yes, with experience, that is important, because during your structural thinking you don’t think about all the eventualities or conditions. Your analysis is wrong, because you have seen that, that and that but not that, which is more or less important. Again this is a trade-off - you are not thinking all your life about all the parameters. You should be able to say; ok, now I have almost all parameters. So when you have experienced, which are the more important parameters, what is the weight of each parameter. Ok, maybe this is a parameter, but maybe not so important.

The one with experience can first see all the parameters, but also he can identify which one is important and which one is not important to solve the problem. The software engineers expressing this conception try on a “higher” level to see all the possible influences, but also are able to see what actually will happen when you go into details further down in the levels. One software engineer expressed this about their work:

S /…/ But, the problem actually in software engineering, the people can know. It is like all knowledge, software engineer, all other technical things - multiple layer of knowledge, we should know, to be able to be a software engineer. That is very general in everything – in medicine you first learn biology, and then actually how hearts work - that based on that. The problem with software, which makes it a little complex, a lot of people begin with the last layer…

I Can you explain the layers?

S The last layer is for example, I gave you the example, what is the “syntax” of a language, what is the “syntax” of C/…/

He concludes to say this about software engineering as a profession and what it is about.

S /…/ it is not rocket science, nothing complex/.../

This also relates to what the other software engineers communicated about making it as simple as possible on a higher so-called abstraction level so that everyone can understand it, and what they are doing to solve the problem. It seems more beneficial for software engineers expressing Conception C that they know and understand the total picture, instead of digging into details. You should understand why to use, for example a programming language, before perhaps how in detail you program.

Still the software engineers expressing this conception are not perfect in identifying everything in the beginning, meaning identifying all the parameters and risks, but they always try to see the whole picture when they are faced with problems within the process, communicated by this software engineer:

I Getting a distance, how do you do that. I understand it, that when your are involved in it, you do it like a routine, but to be able to step back?

S That is the most important part, to be able to take a distance. It is a way of thinking. One of the techniques, in my way of thinking, is when I get inside the problem, and I think a little about that and I am lost, first you should feel that you are lost. So, what I say now is rubbish so stop it. So take your distance, go back and take it again. This for example is systematically very characteristically different between a junior and a senior. Juniors dive into the problem and they should solve it today, and finally they are [lost], and you ask what is the question, that is typical. You have to go out and look at it again.

4.2.3.3 Learning with an intention

Software engineers learn for a specific matter, meaning they have a direct need to be fulfilled, for example they need a specific programming language for this software, so they learn that then. They understand that the learning that takes place has to be in a specific context or situation, being a need for understanding in that part.

S What is very interesting to know is, a lot of things have changed, nowadays you should have a critical way, because another part of software engineering nowadays is learning. Being able to learn, learning fast and learning things, which are needed. So, there is too much information, so finding information, learning new things, you can't learn everything, so I am not able to learn all of these tools, new technical, and I should be able to optimise my way of learning/…/

He continues to say:

S This is very general; I mean you learn a thing if you know what the problem is. You cannot learn things; you cannot learn anything just for the sake of learning. It is just like that. This is my experience - I want to read something to learn the new subject, but what is the problem? Because the new things or tools are the answer to a problem, but what problem will it answer, and that is the problem, a lot people learn the answer, but I mean why? So, why these new tools, match it with what you need.

The last thing he expresses in the interview is:

S When you learn something, you learn it, and as long as you are close to what you have learnt, it is fresh and you are not mastering it. You master things when you have a distance, and you have an overview and you can take a distance from what you know, and put what you learn into its place, that you have with experience. Learning something independent, in an isolated way, that won't help you. Learning is to put it in the context, so this is the difference between someone with experience and someone who has not learnt by experience.

Basically, the software engineers expressing this conception learn for a specific context, for example the current example at hand, as they think it is too much to learn. To be able to understand what you learn, you have to take a distance to what you are doing, and with experience you understand to use what you have learnt for example for other contexts. But, what they look at, which could be said to be somewhat "context free" learning, is to see other ways of solving problems and that is not learning specific programming languages.

S /…/And, so it is good to read other things which you directly are not using, also, so you can get inspired by different ways of solving the problem, so that you see other problem solutions also. /…/

This ties into the next part about interest, where interest is not the interest in specific languages or techniques, but to solve problems.

4.2.3.4 Interest in problem solving

I If you are supposed to think in these abstraction levels that you are talking about, what kind of competences would you need.

S Brain gymnastics.

I Brain gymnastics?!

S Yes, you need to have done it several times, and brain gymnastics, you can say there is a lot of that. You have to think it is good and fun to make such complicated things simple. If you notice a really difficult problem, you should be able to figure out from a big mess… you actually got this part, which is fully done, and this is done, and then we only have this one left [the software engineer is picturing how the complicated thing can be made simple]. And that one we can divide into this, this and that, and then you see - then it is taking shape. But, this part is not looking completely right. Yes, now we have gotten this part to look fine, you can think, but if I move this there, and that one here, and join these two - now we have sorted everything out. That sort of thinking, and to think that is fun, because if you do not think it is fun, then you are not going to do it. Because many think it is an unnecessary fuss to do this, and that you can believe thinking short-term. Thinking short-term it can be that [fuss], but I do not think so.

Being interested in how to make problems as simple as possible is something that we interpreted from the interviews were one aspect in Conception C, as interest in this manner is to think it is fun to analyse and think. To be able to make a mess of things, simple is of key importance, as they will understand how everything fits together.

4.2.3.5 Summary

The interest is to see large complex problems but also to identify what all the parameters and risks are to be able to break it down into a simple problem with the use of so-called abstraction levels is what characterises software engineers working accordingly to Conception C. Getting such a complete overview of the problem as possible before going into details, knowing how the process can be applied to the problem in the most efficient way, avoiding running into unsolvable problems further down in the actual process, but as they identify themselves as humans, who make mistakes, the software engineers expressing this conception, go back up the abstraction levels to get a better overview of what the problems are and what that particular thing or things were that went wrong. Also, an interest in solving complex problems as well as seeing how others solved other tasks can give the software engineers expressing this conception, input and hints how to go about solving their specific problem. To summarise it, the problem is what is in focus and the process here at Iquity is something applied to it.

5 Further analysis and clarification

In this part we will look more closely at the different conceptions to understand what the people expressing these conceptions are about. We will also see the relationship between the three conceptions to be able to understand them more, meaning to get a comprehensive picture of what competence is as a software engineer. Lastly, we will get an understanding of why there is a variation in how software engineers accomplish their respective work/tasks.

5.1 The different conceptions

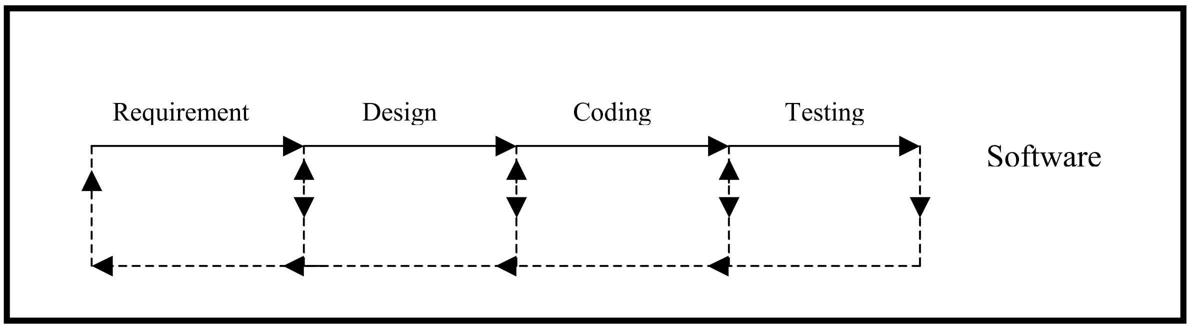
As said previously what we interpreted from the interviews, is that the software engineers are organising but also delimiting the work that they are doing differently, hence we concluded that these differences are how the different competence may be manifested. Putting it in simpler terms is that the way they understand their respective task, is the way they are acting, meaning again organising but also delimiting how they are accomplishing their work. From the interviews and in discussions, we interpreted and concluded that there were three qualitative distinct forms of competence.

To be able to further explain and clarify each conception and in relationship to each other, we will look at what Sandberg (1994) calls a horizon of software engineers work. What we mean by horizon is according to what Sandberg (1994) calls a worker's possible zone of activities. To continue, the horizon can be divided into an inner and an external horizon. The inner horizon is that which is directly conceived and indirectly conceived, meaning that the directly conceived is what is seen, and the indirectly what is hidden or beyond the seen side. The external horizon refers to how the conceptions are related to each other forming a context, a frame of references, which also limits the meaning the work can have for the worker.

5.1.1 Conception A

The software engineers expressing Conception A are in the horizon that the software engineers are organising as well as delimiting their work around the process that is used to solve problems. How this can be explained is that the worker within this conception first starts out with the first step in the process, which is the requirement specification phase, goes on to the design phase, then coding and lastly testing. But, as said before this is not a straight from A to D path, but the software engineers can loop around different steps, for example if they notice that something is wrong or missing in one of the previous stages, like requirements gathering phase, then they loop around to go back to that previous stage, to redo the design if necessary, and then the code, and lastly the testing, in the so-called waterfall principle talked about in the analysis of the interviews. The figure beneath tries to illustrate how it can look, firstly with the different steps, but also how the software engineers can loop back to correct if they did something wrong or forgot something.

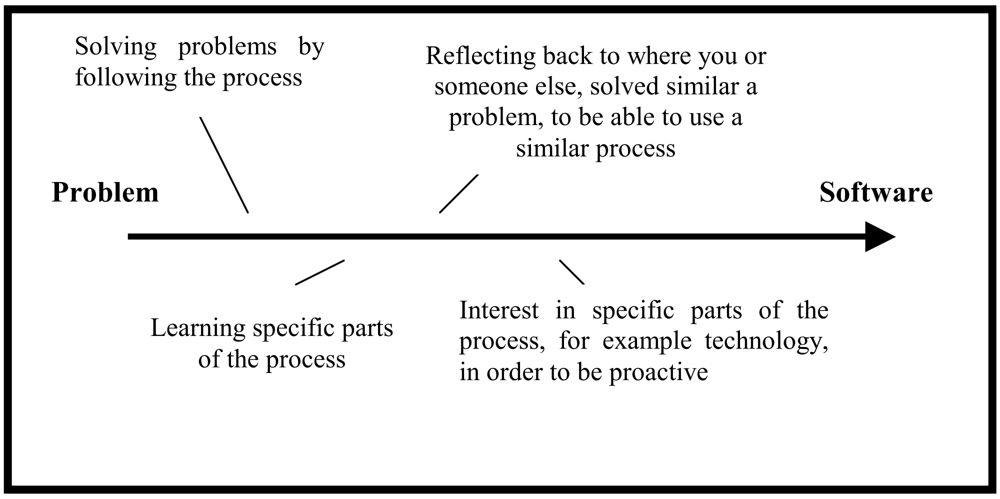
Figure 4 (Own construction)



From the different aspects presented for this conception, one can see that these aspects both have a specific meaning for each conception, but more importantly organises and delimits how the software engineers conceives their work. An understanding of the whole process is seen as important, and not only to be a specific programmer, or designer is seen as sufficient enough to be considered competent.

One can say that software engineers expressing this conception are trying to apply the problem, which they have gotten, to the process. There are a wide variety of problems that the software engineers are facing, and there is not one single problem that is completely identical to the previous, so adapting the problem to the process, is quite general. Also, the problem solving is based upon a specific technology or language that the software engineers know or are using. In Iquity's case, they are using C++ and Delphi with DCOM to link these two program languages, and thus in the problem solving, how one can solve the problem by using these languages. As previously described that "management" at Iquity, in the beginning, found that this process as well as languages and technologies gives the best general solution to the wide variety of programs that they face.

Figure 5 (Own construction)


Solving problems by following the process
Reflecting back to where you or someone else, solved similar a problem, to be able to use a similar process
Problem
Software
Learning specific parts of the process
Interest in specific parts of the process, for example technology, in order to be proactive

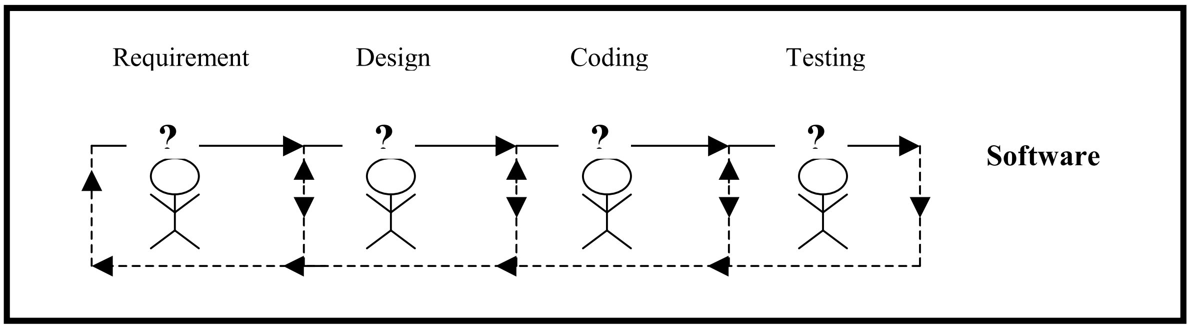
This figure illustrates the different aspects for Conception A, which is both organising, but also delimiting the software engineer's work.

5.1.2 Conception B

The software engineers expressing Conception B are organising and delimiting their work in a "broader" horizon, meaning that they have a broader zone of activities to accomplish their work. As well as organising and delimiting their work around the process as the way to solve, hence create software, they are also looking at the question of why they are doing things the way they are, and how the process can be improved and refined. The goal, it could be said, is to get as good and efficient processes as possible in general for the problems that they will and can face now and in the future.

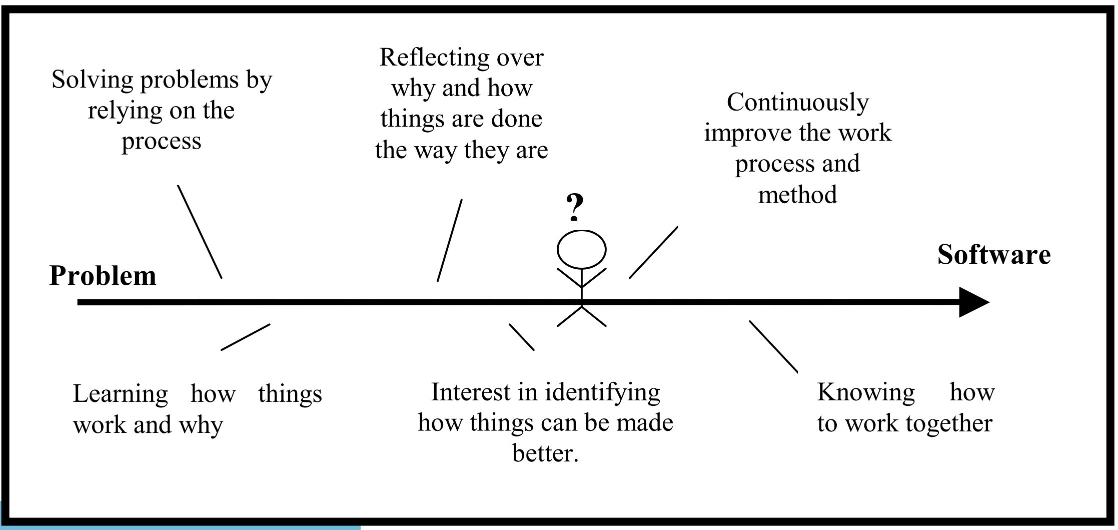
The why and how questions thought about, are a kind of reflection about the whole process but also parts of the process, why they are doing things the way they are, and how they can make it better.

Figure 6 (Own construction)



Also here, there are aspects presented for this conception that both organise and delimit the accomplishing of work. Still the focus is on the parts or the whole process to solve the problem, meaning the problem is applied to the process itself, but what is different, is that a constant reflecting why and how the process or parts of the process are done the way they are done. As a frame of reference for their reflection are other processes and methods used by other organizations, but also there are companies just focusing on selling these kinds of processes and methods. The aspects expressed within this conception also have a specific meaning for this conception.

Figure 7 (Own construction)


Solving problems by relying on the process
Reflecting over why and how things are done the way they are
Continuously improve the work process and method
?
Problem
Software
Learning how things work and why
Interest in identifying how things can be made better.
Knowing how to work together

This above illustrates how there is a constant reflecting back on the process or also parts of the process to see why they are doing things the way they are, and how they can improve and refine the process or parts of it.

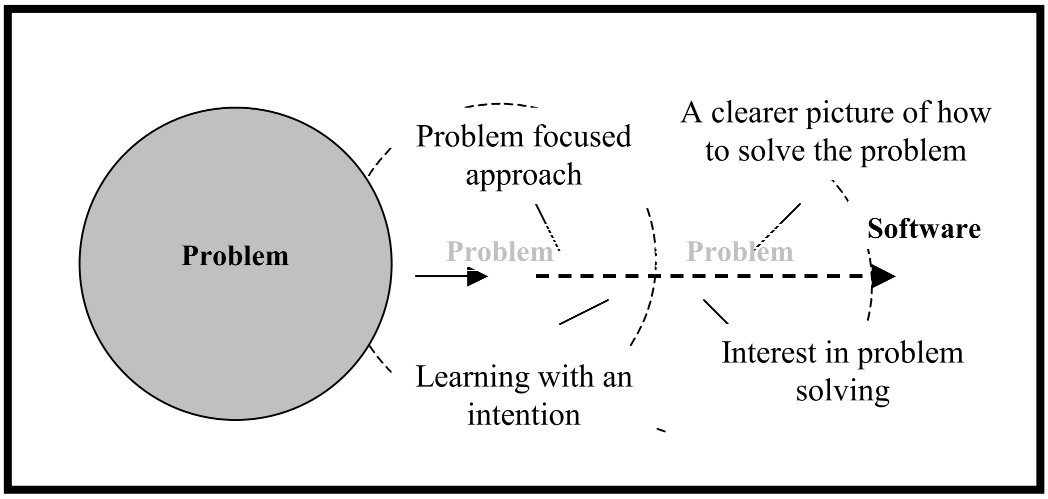
5.1.3 Conception C

The major difference between the software engineers expressing Conception C compared to the others is that there is not a focus and reliance on the process, but instead the focus is on the problem. This could be explained by the fact that the workers in Conception C have a broader horizon, meaning in what zone of activity the worker is conceiving, hence accomplishing his job. But, what is hard to identify, perhaps because of insufficient data expressed within the interviews, is that they, as software engineers expressing Conception A and B, would have applied the problem to the process. Instead, what the software engineers are expressing is that the process is seen more as a guide or a tool to solve the problem, once you have the necessary overview and clear overview of what the problem actually is about. The process is something that they follow, as it is the way you should do it here at Iquity, but there is more an awareness of why also. Questions posed are what, why and how this actually is a problem, and how should we most effectively solve it.

Also, in contrast, but still in connection to the workers expressing Conception B, the so-called process improvement and refinement is done on a continuous basis, meaning that each time they face a new problem, they are adapting the process to the problem itself, in the best possible way. Also, this is not something that we could conclude from the interviews, but in close discussions afterwards with an IT consultant, we speculated that this could be a valid reason. But, still what is common with the other software engineers communicating the other qualitatively different conceptions, is how their conception of work organises and also delimits how they are accomplishing their work, and that is that the problem is focused, with an intention to get as clear an overview as possible to solve the problem as smoothly as possible.

Also, what is even more different is when one looks at the aspects interpreted within Conception C, which organises and delimits how they conceive their work. Still the delimiting is around the problem itself and to get as good understanding and overview as possible before actually using the process. Also, the problem is always in focus throughout the process, as it is used at Iquity, interpreted in the interviews as well as illustrated by the picture beneath.

Figure 8 (Own construction)


Problem
Problem focused approach
A clearer picture of how to solve the problem
Problem
Problem
Software
Learning with an intention
Interest in problem solving

As the problems that the software engineers face are each unique, this is something that the software engineer expressing Conception C is noting, not to say that the other software engineers expressing the other conceptions are not, but software engineers expressing Conception C understand that and because each problem is unique, one has to get as clear a picture as possible of the problem, to be able to make it as simple as possible, then to apply the process to it to get a complete software. Thinking in abstraction levels, constantly analysing and thinking to see and understand all the parameters and risks in the beginning, not getting into detail, is done in order to have a smoother way to the finished software.

5.2 Broader horizon, what does it mean?

What we can see from the previous section is that there is variation between the expressed categories. We could also say that there is variety in how comprehensive a picture they are working according to, which we concluded from these qualitatively different groups interpreted in the interviews. As in Conception A, the software engineers are working following and relying on the process to solve problems, hence develop software, the software engineers expressing Conception B, worked according to the process also, but additionally were thinking and reflecting over why and how they were working in the way they were in the process, in order to see how they could improve and refine it.

In Conception C, software engineers expressed another qualitatively different way of accomplishing work compared to Conceptions A and B, and that is to more focus on the problem and to really understand and get an overview of the problem, in order to understand how the process should be applied to the problem in the most efficient manner. What they communicated as getting an overview was to see it on a higher abstraction level to identify all the parameters influencing and also identify risks that may occur, to solve these before going into the process, even if it for example could be a technology issue. An example of this is not identifying risks early, and this quote illustrates when the software engineers already are in the actual process: "Researchers [at IBM] found that purging an error by the beginning of design, code or unit test allows rework to be done 10 to 100 times less expensively than when it's done in the last part of the process, during unit test or functional test" (Fagan, 1976).

Software engineers expressing Conception C could be said to be using a more sophisticated and more ambitious approach yet simpler, because once they get into the problem, there has been an identification of many parameters and risks, which others expressing Conception A and B hopefully find later in the process. This way of getting it simpler is not an easy task as communicated by this Software engineer expressing Conception C:

I And this simple way, to organise it, it is?

S <u>That is not simple at all. That is very hard. It is very hard to make it [problem] simple, and make things simple. Most do not seem to make it as simple as possible, but there is a lot of mentality to copy. Yes, I have done this before sometimes</u>/…/

5.3 Hierarchical view or not?

To further explain it, we were not completely sure if we could view it as a hierarchy as Sandberg (1994, 1998, 2000) managed to do, but rather we saw that they had a more comprehensive picture, meaning more in C than B, and more in B than A. What we could argue for is that A and C could be seen as dichotomies, as we could *not* identify and believe that a software engineer could work according to both a process focus and reliance and see the problem in focus and process as a guide or tool to solve the problem. This would of course not be true for software engineers being in the so-called transition phase, but still we believe that it still is an either or relationship. We are aware that we are contradicting ourselves, but we could not conclude anything, without feeling too drastic, about the definite answer concerning this. This is what a software engineer expressing Conception A, that we interpreted as being in a transition phase, expressed about a software engineer who expressed Conception C when looking at solutions:

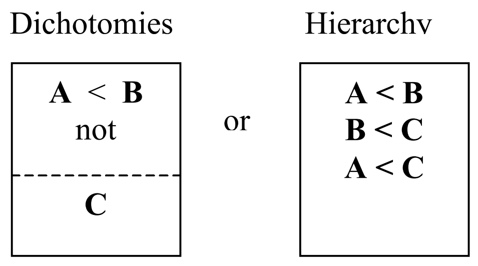
S /.../There are a lot of things that you learn, but just that thing [feeling] is…<u>I feel that it at least is a lot of this “gut-feeling” thing. But, that is different I know, because we work differently down there [among Software engineers].</u> Yes, but a “gut-feeling” is good to get you in the right direction, but you have to be able to motivate it [what you feel]. And that is something that you notice here, because if you cannot motivate something, so even if it is proven right, it isn’t right, because you should be able to motivate it. If not, you are not going to get it through, so you have to learn how to motivate that also. That I know, because I work a little bit differently, meaning I often say *this is how it is* and to motivate it, *no, but why - it is good*. Then you have to think for a while, then come up with therefore, and then it is going to “hold” or not. <u>If you get XX against you, then it is never going to hold</u>.

This quote above also shows that the software engineers that we interpreted as being in a transition phase between Conception A and C, are moving towards how workers expressing Conception C accomplish their work, but have not yet got there, and hence their arguments are still not as well-thought of.

Even if Conception A and C are dichotomies, this was something that we could not relate to Conception B, as it seemed that it was not an either or relationship, meaning then the part of continuously asking why and how in order for process improvement and refinement. As we see it, there is a so-called broader zone of activities for software engineers expressing Conception B than A, because they are also looking at continuous improvements to the process, in addition to reliance and focus on the process.

What we could not interpret, but perhaps speculate on, is that a worker expressing Conception C expressed what Conception B is communicating by asking continuously why and how about the process being used. However, it could still be seen that software engineers that communicated Conception C were making continuous process improvement and refinement for each new problem they face. This means that he should have such a think, that he can find the most optimal way of using the process for each new software development. We could not conclude either of these two speculations, meaning Conception C being a stand-alone phenomenon or being a part of a hierarchy, as we had not gotten that expressed or interpreted from the interviews. But, if it were that the software engineer expressing Conception C, made a conscious or unconscious, process improvement, then we would have gotten a hierarchical variation as Sandberg (1994, 2000) got in his study.

Figure 9 (Own construction)



Going back to software engineers being interpreted for expressing both Conceptions A and C, meaning being in a transition phase, it seems that if we saw it as an hierarchy, then should not a worker go through the Conception B phase before expressing Conception C in accomplishing work? But, at the same time, if workers communicating Conception C is a dichotomy to Conception A, then there could be a so-called change in conception from A to C.

5.4 Explanation for variation in conceptions

But, why then are these software engineers expressing different conceptions. What could be a possible explanation to the variation of competence is to tie it to their educational background and/or work experience. Another thing that we started to think of afterwards, in discussions with software engineers and management is perhaps that the different role that each software engineer has, explains the variation.

As everyone got at least a degree from university or more than 4 years of studying computer science or related programs, we could not directly conclude if it was the educational background that could be the result of this variation in competencies. Even though, one of the software engineers expressing Conception C had the most qualifying academic background, we could not directly conclude that this would be a reason for this variation. But, perhaps this has led to analytical thinking in abstract levels to be able to get a well thought overview.

But, when we then started to look at work experience, we could see some variation linked to this. The software engineers who had worked the longest seemed to be either in the transition phase, meaning between A and C, hence expressing both A and C, or expressing Conception C. What can be seen is that the software engineers with more than four years of work experience could get into this way of thinking and analysing problems. This is what a software engineer said, why he was thinking and analysing, before going into details:

S Yes, it is not completely right either to compare it to these IQ tests, as they are something totally different and have nothing to do with this. Because, in that you only see how to solve puzzles. But here, what can it be, perhaps a course then? If I try to get it down to a course level, there is a course – no! <u>I think that you have to have made enough mistakes in programming to see that this is needed</u> [analysing and thinking in abstraction levels]. I think it is this way. <u>Simply experience, a lot of experience</u>.

To have gone into enough "bumps and bruises" seems to be the most fruitful way for software engineers to change their conception if this is the goal (see section 5.5). So, this could be concluded that it depends on how long you have worked as a software engineer, because it seems logical that the longer you have worked, the more setbacks you have bumped in to. However, what was concluded from Sandberg (1994), and in discussion with a management consultant company, it can be so that even if you have long experience, including that he phases problems in his approach, he could be "stuck" in his conception in accomplishing his work. Also, as said before in section "Hierarchy or not", it could be that a software engineer could change his conception directly from Conception A to Conception C. Also, a person could of course work according to Conception C from the beginning, but that was neither the case in our study or by Sandberg (1994,2000) or Sandberg and Targama (1998).

In discussion with Iquity's management and software engineers, they expressed that the software engineers have mainly two "roles" when accomplishing their work, and that is either that they solve well-defined problems or that they solve non-defined problems. What they mean by well defined problems is that the problems are either very straightforward, meaning that there is not a need for identifying more parameters or risks, hence it can be a pure production like implementation work only. The other so-called role, the non-defined, is that the problems are very unclear, with a lot of parameters and risks that need to be identified and their goal is either to make it well-defined in order for the others, working with well defined problems, to finish up the rest, or that they finish it themselves.

We could speculate, and implicitly sometimes conclude from these discussions, that these roles could be tied to the variation in the conceptions; hence what we interpreted, the three conceptions could be explained by the different roles that the software engineers have when working. An example of this could be that software engineers expressing Conception A and B perhaps mostly work with problems that are well defined. On the other hand, workers expressing Conception C perhaps mostly work with non defined problems, meaning that the problem is very unclear, with a lot of parameters and risks that need to be identified and again their goal is to make it well-defined for the ones in Conception A and B to take over, as almost everything is teamwork.

This is perhaps also linked to years of working as a software engineer or educational background, as the more qualified the management think you are, the more so-called advance tasks you are put in. To tie this to changing one's conception, which will be discussed later, software engineers not put in another situation than just working with well-defined problems, perhaps will never be given the opportunity to change his conception, if management does not perceive him as experienced and talented enough to pursue non-defined problems, as they are seen as more advanced and complex and this will lead to this variation of conceptions. Again, we cannot say that this is the truth, but what we could interpret and speculate from the ones we discussed with said explicitly and implicitly

5.5 Which conception is preferable?

In this section we want to speculate on the different conceptions we have identified and how they can be treated depending on the organisation's desired scenario. Embracing the picture that we interpreted from the interviews, that there are qualitatively three different ways to conceive work as a Software Engineer, which organise and delimit their work, we believe it to be important to have this in consideration when organising work, recruitment and development activities.

As described previously, workers in Conception A solve a problem by applying it to the process, relying on the process to develop the desired software. If we speculate, this can be a way to handle complex and unique tasks such as developing software, having a process or method to reduce uncertainty and complexity. An organisation, by having a process that the workers should work accordingly, can help workers reduce the complexity of the task at hand. The process enables the worker to do his job with reduced uncertainty and at the same time allows management to have some control over the work carried out by the workers. It may also be a way to routinise and standardise quite complex tasks. Also, for newcomers or new graduates, it may be of great help to get going with the job, having a process to rely on when working, sharing a common language and giving a worker a feeling of belonging (see section 5.4)

On the other hand it might hinder innovation and creativity. The question is also if it is fruitful to try to standardise tasks of complex nature. For the worker it may be very limited, hindering further development and resulting in being rather narrow-minded. Looking at the individual level, it might be good for beginners, however it could in the long run result in workers leaving the company as they feel that they are not developing and it is simply not motivating to work in such a way. With experience, workers may challenge the process, consciously or unconsciously and start to change conception, changing from Conception A to B, B to C or another combination. In such a case, it is up to the organisation how rigid they are on working according to the process, does the organisation want to allow adjustments of the process or having workers working in other ways than the process dictated by the organisation.

If we assume that the conceptions identified are hierarchically linked, with Conception C having a broader horizon hence most comprehensive level of competence, the question is if an organisation wants to "optimise" meaning getting all workers into Conception C. Will an organisation benefit from having all workers working in Conception C? Workers in Conception C can be seen as somewhat “visionaries”, working in their own way and not always using the process as it is set up by the company. If the workers meet the deliverables on time, deliver the results they are supposed to deliver it might not be that bad, as they perhaps notice parameters and risks that others do not see in the beginning. The question is rather that you have deliverables, what should be done, but how you do it is up to the individual and in this case it depends on which conception a worker expresses. So, which conception is the preferred depends on what the organisation wants or thinks is the best way. If the organisation believes that workers should use and go through the process as the company sets it up or that they only work with well-defined problems, Conception A or B should be the preferred conceptions, as software engineers working accordingly to Conception C are not really necessary. If it does not matter how the workers reach their deliverables or that most problems are non-defined, Conception C could be seen as preferred, allowing workers to reach their deliverables in their own best way, not necessarily following the process step by step, but also identifying parameters and risks more “smoothly”, meaning not in the end when one has to go all the way back to redo everything. What to remember is that Software Engineers expressing Conception C are still capable, as they do here at Iquity, of using the process set up, but it is just seen as help or a tool.

However, if a worker in Conception C is too much of a “visionary”, if he does not deliver what he is supposed to and works too much in his own way, it could be troublesome for the company. This could result in unstructured and uncontrolled work situations and his co-workers would probably view him as incompetent as he cannot fulfil the most basic requirements.

On the other hand, having all workers working in Conception C can also be very fruitful, finding innovative ways to solve new problems since the process does not limit them, because they look outside the process. On an individual basis, we could speculate that some workers strive for being independent of work processes and procedures and having a large “action space”. At the same time some workers might prefer having a process to follow giving them a certain degree of stability and reducing uncertainty. On an organisational level, management might feel to some degree that they lose control by having too many workers working in Conception C, but at the same time it facilitates innovation and creativity.

But, in the end, still everything comes down to the fact that it is only the individual himself that can “decide” whether or not he wants or is able to change conception; assuming that he is aware of his present conception and understands that there are qualitatively other ways to accomplish work that perhaps could be better. An example could be to change from Conception B to C, that the worker understands that moving to Conception C does not mean that you abandon the process. But that there could be other ways to solve a problem or use the process, not just improving the existing process as workers in Conception B express.

5.6 Shift in conception as the goal

Depending on scenarios, it will vary in what both the organisation and the individuals want. But as we talk about it (in section 5.2) that Conception C is expressing a broader horizon than B, and B would work accordingly to a broader horizon than A, we felt it would be natural for the thesis to discuss how one could shift conceptions. A logical first thought could be to get all workers to get to Conception C. When we discussed this with an IT consultant, he saw it as “optimal” for the organization, to have workers conceiving their work as in Conception C. We on the other hand think it is very individual from organization to organization, but also from software engineer to software engineer as was our intention to show with the scenarios, what could be preferred. But, again if the intention is to get the software engineers to shift conception from A to B, B to C, or even A to C, it is argued and proven (Sandberg 1994, Sandberg & Targama 1998) that the so-called rationalistic method, the attribute based perspective, would not get the software engineers to shift competencies. It would only reinforce the current way of accomplishing one’s work. It is still important to remember that knowledge and skills spread through courses, school, on the job-training are necessary, as it is what precedes the competence or conception, like a kind of “raw material”. “It is the workers’ ways of conceiving work that make up, form, and organise their knowledge and skills into distinctive competence in performing their work.” (Sandberg, 2000)

But, how then could a worker change his way of accomplishing his work. We actually, from the interviews, (in section 5.4) have a software engineer that expressed how he had himself unconsciously changed his conception and he did that by incurring so many bumps and bruises that he seemingly forced himself to change his way of performing his job. This was done more or less on a forced basis; therefore we would like to discuss how this could be done hopefully less “painfully” and more systematically.

The first step as we see it, is to understand that the so-called reality around us is individually- (Husserl, 1970) and/or socially constructed (Berger & Luckmann, 1966) (discussed in section: 3.3). One has to understand that one is able to “deconstruct” one’s way of accomplishing a task, and prescribe another meaning to the event, to be able to accomplish one’s work in a qualitatively different manner.

The second step is to understand one’s own competence/conception, meaning a form of self-reflection (Sandberg, 1994, 2000, Sandberg & Targama 1998), to understand how you are working now and to be able to understand its limitations. In addition, the software engineer needs to be challenged, so when he reconstructs his competence, he is accomplishing his work in a qualitatively different manner. As we can see from the software engineer changing his way of accomplishing his work, he noticed that it was too “painful” for him to go on like this, hence he deconstructed his conception, challenged it, and reconstructed it into a new conception, namely Conception C. This he did in an unconscious and conscious mode, but we believe that if we could make this more visual and conscious for the other software engineers, it can be facilitated and more effective.

Prescribing a change in this way by making it more conscious, both that a conception or understanding of one’s work can be deconstructed-reconstructed, but also make the worker himself aware of how he is organising and delimiting his work, we hopefully can faciliate a change in conception. It is logical for workers, as Sandberg and Targama (1998) state, that in the acual performace of the work you are doing, all our attention is directed towards the work or task, and not towards our understanding of our work. They are also arguing that if a person never reflects over his way of accomplishing his work, he is never able to change his understanding, hence start to see other alternative ways of understanding his job.

A suggestion for how the organisation itself could help the software engineers to change their competence or understanding of their respective work, could be to position them in a situation, which is described above, to let them challenge their understanding. This is simpler said than done, as we discussed it with an IT consultant, and he could not see any good methods or ways to create such "situations." A start could be (discussed in section 5.4) to place the Software Engineers in situations where they face non-defined problems.

In addition, in our reflections and discussions, we came to the conclusion that perhaps, it could be done by provoking or constructing a so-called "critical-incident". This term was used by Flanagan (1954) in the 1940s to avoid airplance accidents, but in our case we would not want them directly to find out ways on how to prevent them in future, but rather have this as a "challenge", making them reflect over their current way of organising and delimiting their respective jobs. Still, we saw it as very hard to provoke something like this, as we would not know what kind of critical incidents would be suitable for this purpose.

Also, a natural step would be that in discussions and interactions with software engineers expressing another conception, the person would get insights and understand that the way he is accomplishing his work could be done in a qualitatively different manner, meaning that mistakes that he had made in the last project, or bumped into painfully during the process, could have been adressed or been less "painful". But, it is not to see this preferable conception with aspects as attributes that you just can learn as you collect information using a rationalistic approach. What we wrote before, is that the present conception will not only organise and delimit the way you are working, but also what and how you are learning, hence it is said by Sandberg (1994) and Sandberga & Targama (1998) that it will just enhance the current way of accomplishing one's work

But, maybe the only way to actally let them change their way of accomplishing their respective work, is to make them aware of their own conception, so that they in a working situation when they face a problem, can challenge their own understanding, and hopfully reconstruct their conception into a so-called qualitatively different one. This would not be so different as what they are doing now in their daily situational learning.

Important to remember is that, even though, there can be a goal for the individual to change his conception, by deconstrucing-reconstructing and by self-reflection, but still it perhaps is not enough, because the person is not able to change his way of accomplishing his work. The reason why a worker cannot change to another conception is something we cannot predict, but perhaps he is content with the work situation he is currently in, and is not challenged to understand tasks in a different matter. An example would be if the software engineer, mentioned before, did not interpret the bumps and bruises as painful, he would perhaps not have challenged his own way of working. We also believe that this is not something done overnight, but will probably take time, but this will vary from software engineer to software engineer. Some, it can probably take a whole lifetime, while others can do it faster.

Again this change in conception, meaning the way a software engineer accomplishes his work has a lot to do what the software engineer wants to do himself as well as the goal of the organisation as a whole, looking at the scenarios. But, it is important that this change in competence could happen unconsciously, but this would not be noticed if he did not know how he worked before, meaning according to which conception, and how he works now. Another important part is that we identified three qualitatively different ways of accomplishing work, but there could perhaps be more groups if we were to interview more software engineers at other organisations, either conceptions that would be interpreted as before Conception A, or between the various conceptions, or also another conception after Conception C, if we were to hierarchially classify them in how a broad horizon they are expressing.

6 Comments and Conclusions

In the beginning of this thesis we concluded that competence is a crucial issue in today's competitive marketplace. Main issues amongst organisations today are attracting and developing competent employees in their quest to gain a competitive advantage. In order to manage competence in an organisation, we believe the first step is to identify what constitutes competence. At first, our aim in this study was simply to identify competence as a Software Engineer at Iquity Systems. During our search for a fruitful method to identify competence, we realised that there were a lot of limitations in contemporary management practices within this field, the attribute-based approach. The limitations we found were not easily explained but became quite clear and visible on the meta-theoretical level, the ontological and epistemological assumptions underlying these approaches.

Due to the limitations we saw, a shift in perspective directed us into an interpretative approach, phenomenography, where it is argued that in our case, a Software Engineer's understanding for his work organises and delimits his way of accomplishing work, consequently the competence as a Software Engineer. This also resulted in a modified aim of the investigation, identifying what constitutes competence as a Software Engineer by taking the Software Engineers' conception of their work as the point of departure using a phenomenographic approach. Abandoning the rationalistic approach of viewing competence as an attribute-based phenomenon and we found that embracing the interpretative approach put a lot of demands on us as researchers.

During our research, we found our thoughts to be in a continuous battle, switching back and forth between the rationalistic and the interpretative view of competence. Throughout this journey we could say that we have been in a so-called transition phase, in the process of changing conception of the phenomenon competence. At the same time as it has been fun, challenging and fruitful, it has also been really intellectually challenging, time consuming and complex. However, we believe that we have with this thesis contributed to the field of research within the area of competence.

Firstly, we believe that we have shown a potential other way to identify and view competence in an organisation. Using a phenomenographic approach in identifying competence has enabled us to take the context into consideration and thus described competence in a more direct way than using an attribute-based method. The context meaning that we have based the description of competence on the interviews where the workers themselves have described their experience and interpretations of what competence is as a Software Engineer.

Secondly, and most interestingly, we have shown that there are three qualitatively different ways of accomplishing work amongst the Software Engineers at Iquity Systems, three different conceptions of competence as a Software Engineer. What is important to notice is not really in what conception category each worker has been positioned, but rather it is what the Conceptions themselves represent that we interpreted as three qualitatively different competencies that is of importance.

Thirdly, taking the above into consideration puts new demands on recruiting and competence development. Much of recruiting and competence development activities in most organisations today are based on the more traditional view of competence, the attribute-based perspective. Embracing the idea of seeing competence as the conception of work a worker has implies that in order to develop competence a shift in conception is needed. We could also differ between developing competence and reinforcing present competence, present conception of work. It all depends on what management and workers see as a goal, which conception is most preferable. We will not speculate further in this area since our aim in this investigation was only to identify competence. However, we feel it to be of great importance to mention how the findings will affect the management of competence, such as recruiting and competence development.

Fourthly, we would like to mention our view of using this method in identifying competence in an organisation. We feel that the view of competence that we have shown in our thesis gives a more comprehensive picture of the concept of competence than the attribute-based view. It describes competence in a more direct way and it seems now after we have done interpretative research on competence, as this is the most natural and most logical way to identify and view competence.

However, one question remains, something that would be really interesting to further investigate, if this method can be used in a systematic manner. Our work has been very heavy, time-consuming and many times frustrating. If an organisation were to identify competence in the entire company using this method, it would most surely require a lot of full-time effort and struggling in the analysis part and transcribing interviews. So, has an organisation the time and will to conduct such a project if they know how much time and effort we have put into this thesis? And, would we be ready to recommend the use of this method?

Those are really tough questions to answer since as we mentioned before, now that we have conducted a phenomenographic approach in identifying competence, we do believe this way to be the most beneficial and logical way to identify and view competence. At the same time we are experienced with the facts that it is heavy work.

One has to remember that research using phenomenography in business and organisations is very limited and hence in its “infancy” stage. As it is growing we hope more systematic ways of using the method and systematic usage of the results will gradually be developed. We do believe that we have contributed to the field and hope it will be beneficial for others engaging in phenomenographic studies in identifying competence.

7 References

Articles and Books

Berger, L., & Luckmann, T. (1966) *The Social Construction of Reality*. Penguin, Harmondsworth, England

Berztiss, A. (1996) *Software Methods for Business Reengineering,* Springer-Verlag, New York

Blackler, F. (1995) Knowledge, Knowledge Work and Organzations: An Overview and Interpretation, *Organizational studies*, Vol. 16, Iss. 6

Brown, J. S., & Duguid, P**.** (1998) Organizing Knowledge, *California Management Review*, Berkeley, Vol. 40, Iss. 3, Spring

Brown, J. S., & Duguid, P. (1991) Organizational Learning and Communities-of-Practice: Toward a Unified View of Working, Learning and Innovation, *Organizational Science*, Vol.2, No.1

Cook, S. D. N., & Brown J. S. (1999) Bridging Epistemologies: the Generative Dance Between Organizational Knowledge and Organizational Knowing. *Organizational Science*, Vol. 10, No. 4

Dahlgren, LO. (1997) *Learning Conceptions and Outcomes*, in Marton, F., Hounsell, D., and Entwistle, N. *The Experience of Learning*, Scottish academic press, Edinburgh

Entwistle, N. (1997) *Contrasting Perspectives on Learning*, in Marton, F., Hounsell, D., and Entwistle, N. *The Experience of Learning*, Scottish academic press, Edinburgh

Fagan, M. E. (1976) Design and Code Inspection to Reduce Errors in Program Development, *IBM Systems Journal*, Vol. 15, no. 3, 182-211

Flanagan, JC. (1954) The Critical Incident Technique. Psycho Bull; 51: 327-58

Gael, S. (1988) *The Job Analysis Handbook for Business, Industry, and Government*, Vol, I, II. John Wiley & Sons, New York

Husserl, E. (1965) *Phenomenology and the Crisis of Philosophy*. Harper and Row, New York

Husserl, E. (1970) *Logical Investigation*, Vol 1,2, London

Marton, F., Hounsell, D., and Entwistle, N. (1997) *The Experience of Learning*, Scottish academic press, Edinburgh

Marton, F. (1986) Phenomenography – A Research Approach to Investigating Different Understandings of Reality, *Journal of Thought*, Vol. 21, No. 1

Marton, F. (1981) Phenomenography: Describing Conceptions of the World Around Us, *Instructional Science*, 10, 177-200

McConnell, S. (1993) *Rapid Development*, Microsoft Press, USA

Norén, L. (1995) *Tolkande Företagsekonomisk Forskning : En Metodbok*, Studentlitteratur, Lund

Orr, J. (1990) *Talking About Machines: An Ethnography of a Modern Job,* Ph.D Thesis, in Brown, J. S., & Duguid, P. (1991) Organizational Learning and Communities-of-Practice: Toward a Unified View of Working, Learning and Innovation, *Organizational Science*

Polanyi, M. (1966) *The Tacit Dimension*, Doubleday, New York

Polanyi, M. (1977) *The Tacit Dimension*, in Laurence Prusak (ed). Knowledge in Organizations, Butterworth – Heinemann, Newton

Prahalad, C. K., & Hamel, G. (1990) The Core Competence of the Corporation. *Harvard Business Review*, 3, 79-91

Pramling, I (1983) *The Child's Conception of Learning*, Acta Universitatis Gothoburgensis, Göteborg

Ferris, G. R., Rowland, M., and Buckley, R. M. (1990) *Human Resource Management: Perspectives and Issues,* Allyn & Bacon, Boston

Sandberg, J. (2000) Understanding Human Comptence at Work: An Interpretative Approach, *Academy of Management Journal*, Vol. 43, No. 1

Sandberg, J. & Targama, A. (1998) *Ledning och förståelse,* Studentlitteratur, Lund

Sandberg, J. (1994) *Human competence at work: An interpretative approach*. Bas, Göteborg

Senge, P. (1999) *The Dance of Change : the Challenges of Sustaining Momentum in Learning Organizations*, Doubleday/Currency, New York

Sherman, R. R., & Webb, R. B. (1986) Qualitative Research, *Journal of Thought,* Vol. 21, No.3, Fall

Säljö, R. (1982) *Learning and Understanding: A Study of Differences in Constructing Meaning from a Text*, Acta Universitatis Gothoburgensis, Göteborg

Targama, A., & Diedrich, A. (2000) Towards a Generative Theory of Knowledge and Its Implications of Knowledge Management, Göteborg

Teigland, R., & Timlon, J. (1998) Knowledge Dissemination and Communities of Practice: A Case Study of Software Programmers in the Internet Industry, Work-in-progress, Stockholm School of Economics

Theman, J. (1983) *Uppfatting av politisk makt (Conception of Political Power),* Acta Universitatis Gothoburgensis, Göteborg

Van Grogh, G., & Roos, J. (1995) *Organizational Epistemology*, MacMillan Press, London

Weick, K. E. (1979) *The Social Psychology of Organizing,* Addison-Wesley, Massachusetts

Wenestam, C. G. (1982) *Children's Reactions to the Word Death,* Göteborg

Wenger, E. (1998) *Communities of Practice: Learning, Meaning and Identity,* Cambridge University Press, Cambridge

Internet

Iquity Systems Inc. (2000) www.iquity.com

Carnegie Mellon University: (2000) Software Engineering Institute. Capability Maturity Model® (SW-CMM®) for Software, www.sei.cmu.edu/cmm/cmm.html

Personal Communication

Dickson, P. (2000) Discussions on various occasions, KnowIT, Fall

Targama, A. (2000) Discussions on various occasions, Gothenburg School of Economics and Commercial Law, Fall

Wästfelt, L., & Elliot, K. (2000) Discussions on various occasions, ConcoursCepro, Fall

Appendix 1

Identifying competence by using Phenomenography

The word "phenomenography" has its etymological roots in Greek "phainomenon" and "graphein", i. e. "appearance" and "description". The combination of these two words makes "phenomenography" a description of appearances. - Amedeo Giorgi, 1991-

By using phenomenography when identifying competence at Iquity, we are not trying to find an objective truth. Nor do we want to use the result to generalise, saying that this is the case amongst Software Engineers in general, even though one could speculate in if this could be the case. We wanted to obtain the Software Engineers' description of their experiences of their work as a potential way to explain what constitutes Software Engineers' competence at Iquity Systems.

Our first step towards obtaining the Software Engineers' conception of their work was to present ourselves, explain our interest in competence and to present our ideas about competence during one of their regular Monday meetings. This was done to establish a contact with them and also an opportunity for questions and objections. After the meeting we talked to most of the Software Engineers to get to know them better and to establish trust between them and us.

Obtaining data

Obtaining conceptions can be done in several ways. Wenestam (1982) analysed drawings to reveal children's conception of death. Observations can also be used, as a method to reveal individual's conceptions of a phenomenon. However, interviews have been the primary method of phenomenographic data collection (Marton, 1986). By using interviews, the researcher's intention is to obtain the interviewee's description and experience of a given phenomenon.

We decided to use interviews, since we saw it as hard to use any other method. For example, if we had used observations, we would have needed to get the Software Engineers to "think aloud" when working, which we saw as quite a lengthy procedure. Interviews seemed to be the best way to get their descriptions of their experience in working as a Software Engineer.

Selecting interview participants

At first, we wanted to interview all of the eleven Software Engineers at Iquity, but two of them did not want to participate, which we respected without questioning them, and one of them did not have time to participate. All in all, we booked interviews with eight of the Software Engineers.

Interviewing

In phenomenography, since the goal is to obtain the interviewee's description of a certain phenomenon, it is important that the questions are as open-ended as possible, to get the interviewee to choose the dimensions of the question they want to answer. This is important since we want to get the interviewee's relevance structure of a given phenomenon, not leading the interviewee in any direction. The questions are then followed up by follow-up questions, which depend on the answers the interviewee gives. This means that different interviews can lead to different dimensions of a phenomenon discussed in the interviews.

The interview method that we used could be compared with what Entwistle (1997) would call, a semi-structural interviewing method. Entwistle says that this style is rather different from the research interviews usually recommended. Furthermore, the great advantage is that this interviewing style will be developed as a natural conversation and discussion, but still within a pre-determined framework (Marton, 1997). Also, if the answers we got were not as comprehensive as we felt they could be, follow-up questions were asked for clarification, elaboration and some examples of real-life events and examples of similarities to other profession and events. As for Dahlgren's (1997) avoidance of giving clues about the desired outcome, we felt that we could not give clues, as we did not have a desired outcome, except getting as thorough descriptions of their interpretation of their respective experiences as possible. Still, we are aware that our subjective preferences had an impact, but not in the way that we had a specific goal, how we wanted them to answer.

Another important issue during interviewing is that the interviewer strives to hold back his known theories and prejudices during the interviews so the interviewer does not influence the interviewee in any way.

When we had formulated our questions, we conducted a pilot interview, to see if the questions were good and to get the feel of asking relevant follow-up questions and to test ourselves in being neutral and holding back prejudices.

We conducted the interviews in a "neutral" room. We started by telling the interviewee what our intention was with the interviews and made sure that the interviewee understood what the interview was all about, reminded them that we wanted their experience and description of what working as a Software Engineer is like. We constantly reminded ourselves to hold back prejudices and not ask leading questions. All interviews where taped using two tape recorders, since it was vital for us to get all the information as clear as possible, so we could get as extensive information of their descriptions as possible. Since non-verbal language is not captured on the tape recorder, it felt even more important to get as clear information as possible of their descriptions.

Analysing the data

When we had conducted all the interviews, we started the analysis of the data, searching for essential aspects of work from descriptions of the Software Engineers' ways of conceiving their work. The first step in this process was to transcribe all the interviews.

Writing transcripts

Using the tape recorders, we wrote down all the interviews word by word. This was really time consuming and difficult, since the spoken word differs quite much from the written word. Sometimes the interviewee expressed himself in a way that was hard to understand. Consequently, we had to listen back and forth several times before we could figure out what he was talking about. There is a risk in missing out on words or even statements. When writing down the interviews from the tape recorder we also miss out on the tone of voice. Since we ourselves had conducted the interviews, written the transcripts and analysed them, we feel that to some degree we could remember some critical issues regarding the interviewee's tone of voice, state of mind etc.

Getting an overview of the transcripts

We started the analysis by glancing through each transcript three to four times just to get an overall picture of them. This gave us an overview of the transcripts and we managed to roughly categorise them in various groups. Here we also basically looked at what the Software Engineers conceived as work and categorised them accordingly.

Looking for statements interesting for competence

Next step was to search for statements that seemed to be of interest for finding essential aspects of work. We selected quotes that were interesting and also looked at the context where the quotes were stated. Then we classified the quotes in terms of the context from where they were taken. By doing this, we narrowed down the material and could sort out the information that we found to be relevant. We also got groups that were based on the meaning that lay behind the statements; three groups were established with two borderline cases, which we were not sure where to place.

Looking at the meaning behind the statements

We shifted the focus from finding statements to looking at the relevance of the groups that we had identified based on the context, the meaning behind the statement. We looked at the meaning embedded in the quotes and looked for similarities and differences between the ways in which the work appeared to the interviewees. In this stage, we looked at from which interview the quote had been taken and in which "pool of meaning" the quotes had been categorised. Doing this, we could bring together statements on the basis of their similarities and differentiate categories based on their differences.

Categorising into three conceptions

Finally we could see three qualitatively different groups expressing three different conceptions of working as a Software Engineer. These three categories demonstrate three different core meanings along with attributes seen as important in relevance to their conception. Still, we had two persons that expressed two conceptions and we felt it to be really hard to put them in one of the groups, borderline cases. This might be explained by them being in a "transition stage", in the phase of changing understanding from one qualitative way of understanding to another, thus communicating two conceptions. If we speculated why it was this way, with the help of Targama, we could perhaps imagine that they are in a transition phase between different conceptions. In the material one can implicitly or explicitly interpret that the person is somewhat describing experiences in two or more different conceptions. But, what one can read out, in the interview, is that there is uncertainty where the person really tries to explain their interpretation of their experiences, and the answer could be that they are in a transition phase between two conception categories.

We could perhaps have made a separate category for this, but it is rather speculative and perhaps there are other dependencies. The main importance, still, is not in which respective category we identify the software engineer's description as individuals should be, but instead the importance is to see how they conceive their work differently.

Outcome space

We felt it to be really hard to decide how we should present our findings. One extreme could be to just present the groups and explain what characterises them. The other extreme could be to include all interview transcripts in the thesis. We decided to choose a middle way, to show quotes that we felt could explain what we meant by our classifications.

Validity and Reliability

Since a phenomenographic approach is more of a discovery procedure rather than a measurement procedure, it seems wrong to use traditional criteria in measuring validity and reliability. It seems hard to replicate such a discovery, however, the researcher must communicate the "outcome space" in such a way that other researchers could recognise instances of the different ways of experiencing the phenomenon in question. The most important for us during our research has been to be as faithful as possible towards our material.

Still, what to remember is that our results from analysing the interviews are not something that could be called an objective truth, but rather our interpretation of the interviews, which we analysed being as faithful to the material as possible.

Throughout our investigation we have tried our best to withhold own theories and pre-understanding of the work as a Software Engineer. The main question for validity in our case is to describe the Software Engineers' conceptions as faithfully as possible. During the interviews, we did not have a lot of questions that the interviewee just should answer, we made sure that a dialogue was established, thus generating descriptions of the Software Engineers' experience of their work.

When we had generated the three conceptions, we asked one of the Software Engineers that was not interviewed if our groups felt reasonable, which he agreed they did, and also asked him to place the interviewees in the categories he felt they fitted. To our contentment, he directly placed six of the eight interviewees as we had placed them. We also asked one of the interviewees if he thought the groups that we had come up with were reasonable, also he felt that this was something that was relevant and made sense. We asked him to place the other interviewees in the group he felt was appropriate. He placed four of the interviewees directly as we had, the other three he had a hard time to place, two of them were the ones we had a hard time to place ourselves, that we believed to be in a sort of transition phase.

In the case of reliability it is hard to say that any researcher can come to identical results as we did under the same circumstances. As mentioned before, it is discovery procedure rather than a measurement procedure. However, our challenge has been to present the descriptions in a way that other researchers can see as reasonable. During our analysis, we often sat individually to see if we came to a somewhat similar interpretation. Most often we did, with some differences but mostly similarities. During the investigation we have tried our hardest to not be biased or look for answers to justify conclusions that we might feel personally as reasonable. Again, we have constantly tried our utmost to be as faithful as possible to the material we collected.

Methodological difficulties using the phenomenographic approach

Constantly during our investigation we have tried our utmost to be as faithful as possible to the Software Engineers' own experience of their reality. However, it is hard to withhold one's own interpretations and theories all the time, to always have a blank piece of paper. Most surely, in some occasions we have been coloured by our own interpretations, but we cannot pinpoint a certain situation where it may have occurred. The most important thing to remember is that it is not a question of being completely objective, but to be aware of the fact that our own interpretations in some sense influence us. It is a question of being aware of this and to try to have in mind that we do not want to reach an objective truth, rather to describe the Software Engineers' experience and what and how they conceive their work.

Of course, it is easier to conduct a quantitative investigation, where measuring and drawing scientific conclusions is quite straightforward. However, using a qualitative approach, phenomenography has been really challenging and fun, and sometimes really frustrating and found ourselves hanging on to rationalistic behaviour.

One could say that we have during this journey been in a transition phase between rationalistic thinking and interpretative thinking, pushing ourselves towards the interpretative approach. We were quite easily influenced and convinced that the phenomenological base underlying the interpretative research tradition that worker and work are internally related was a more direct and fruitful way of describing competence as a Software Engineer.

Nevertheless, going from one understanding to another is a fundamental change that is an ongoing and lengthy process, requiring reflection and challenging one's present understanding. Since we have been brought up in the traditional Western school system with a lot of rationalistic influences underlying the view of education, one could say that we ourselves are in the process of changing conception. From our present/former dualistic ontology and objectivistic epistemology view to a qualitatively different one, namely a phenomenological view where person and world are inextricably related to each other.